п11. Член-данные класса - объекты другого класса: агрегированные классы

Член-данное какого-либо класса может быть переменной или указателем, тип которых - другой известный на данный момент класс. Такой класс называется ***агрегированным.***

Рассмотрим пример

```
class A { int x, y;
          public:
         A(int xx = 0){ x = y = xx; }
   // конструктор с аргументом по умолчанию
  A( int xx, int yy){ x = xx; y = yy;}
  void Print(){cout << '\n' << x << '  '<<y;}
  int Getx() { return x; }
  int Gety() { return y; }
};
```

```
class B{  // агрегированный класс
          A a; // объект класса A
          int z;
          public:
          B(){ z = 0;}
         B(int, int, int  );
         void Print();
         };
```

<u>Главное правило</u> при использовании объектов другого класса: КОНСТРУКТОР ИСПОЛЬЗУЮЩЕГО КЛАССА ОТВЕЧАЕТ ЗА ИНИЦИАЛИЗАЦИЮ ЧЛЕН-ДАННЫХ ИСПОЛЬЗУЕМОГО КЛАССА!

Конструктор класса B

Определяется он так:

B(int b, int c, int d) **: a(b, c), z(d)**

список инициализации

{ }

Собственные член-данные можно инициализировать и внутри { }

B(int b, int c, int d) **: a(b, c)** { **z = d**;}

Конструкторы по умолчанию

Заметим, что если в классе A есть *конструктор по умолчанию*, то он будет работать, даже если он явно не указан в списке инициализации конструктора класса B. В нашем примере при работе конструктора **B(){ z = 0;}** будет работать и конструктор
A(int xx = 0).

Порядок работы конструкторов при объявлении объекта класса **B**:

сначала в A,

затем в B.

Агрегирование по указателю

Несколько сложнее выглядит конструктор агрегированного класса, если член-данное в нем - **указатель** на объект другого класса, и он определяет **массив**.

В этом случае конструктор должен **брать память** в динамической области и **инициализировать элементы** массива.

Списком инициализации здесь не обойтись.

Рассмотрим на примере, как в этом случае можно определить конструктор агрегированного класса.

Пусть имеется **класс Array** – числовой массив.

```
class Array { int *a, n;
                public:
              Array(int nn = 1, int t =0 );
/* nn - размер, t!=0 – инициализировать массив
 случайными числами */
              int & operator [ ](int);
friend ostream & operator <<(ostream &, Array&);
                …
};
```

Используем его для определения класса **Matrix:** числовая матрица, рассматривая её как массив массивов .

Агрегирование по указателю

class Matrix

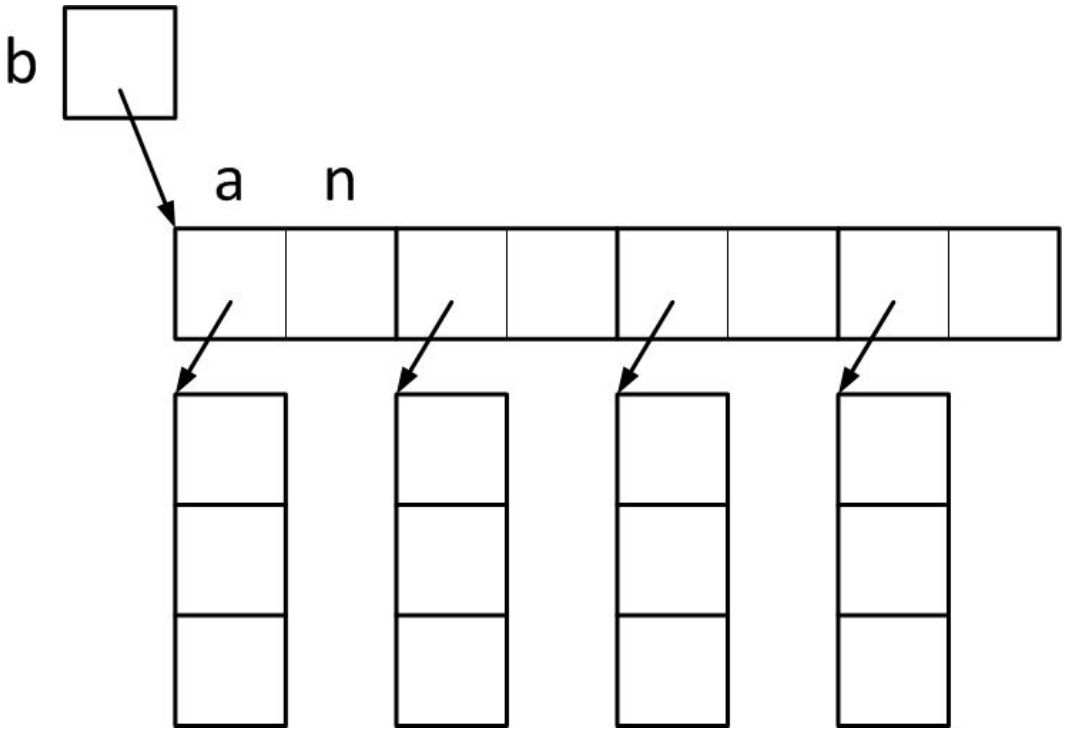
```
class Matrix { Array *b; int m, n;
                public:
                Matrix(int m1 = 1, int n1 = 1, int t = 0);
              /* t !=0 – создание элементов матрицы
                 случайным образом */
  Array & operator [ ](int i)
  // возвращает ссылку на i-ую строку матрицы
  { if ( i<0 || i>=m) throw ”Номер строки неверен!”;
    return a[i];
   }
  void Show();  //  вывод матрицы
... };
```

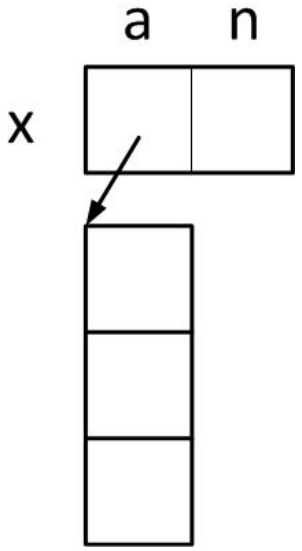
Конструктор

Array(int nn = 10, int t =0);

```
Matrix :: Matrix(int m1, int n1, int t)
{ b = new Array[ m = m1]; /* Так как в этом случае
 используется только конструктор по умолчанию
 класса Array, то такая инициализация
 строк может не подойти. Поэтому  определить
 строки длиной n можно таким образом:   */
n = n1;
for ( int i = 0; i<m; i++)
{ Array x(n, t);  /* создается массив – строка
    матрицы,  t – инициализация */
   b[i] = x;       /* работает перегруженная
    операция ‘=’  класса Array */
}     }
```

Конструктор






```
b = new Array[ m = m1]; n = n1;
for ( int i = 0; i<m; i++)
{ Array x(n, t);
  b[i] = x;}
```

Пример использования

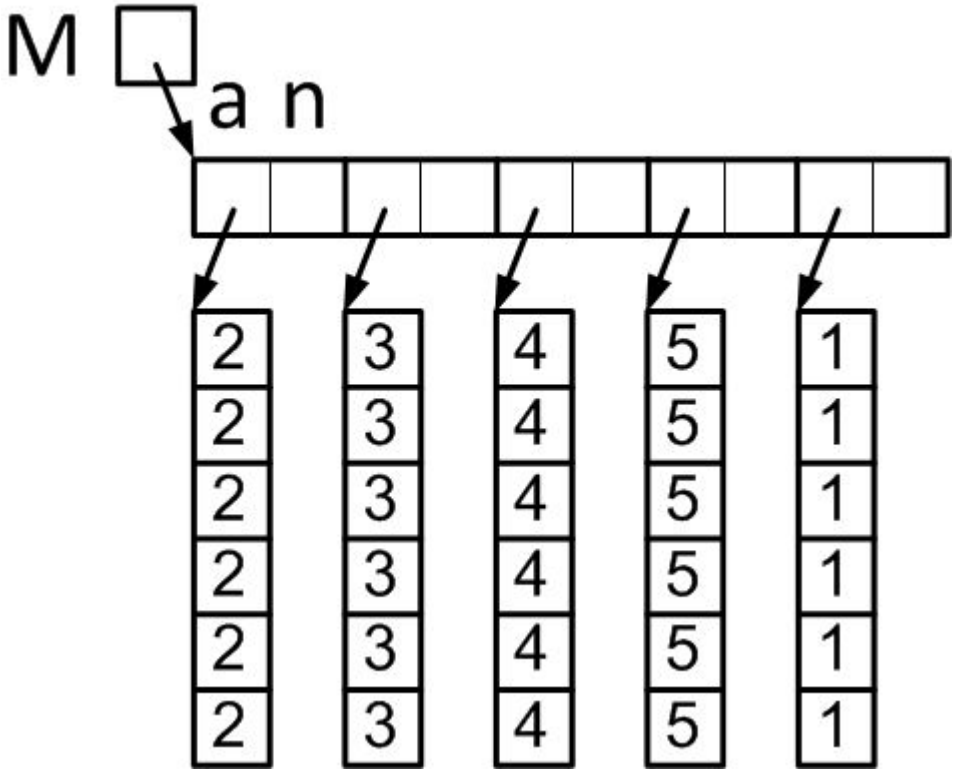
```
void  main()
{ Matrix c(4, 5,1);
   c.Show();
   c[1][1] =100;
/* Работают две перегруженные операции [ ]:
   первая – в классе   Matrix,  вторая – в классе
    Array */
  cout << c[0];
 /* вывод строки с номером 0 по операции
    потокового вывода,  определенной для класса
    Array */
}
```

Задача

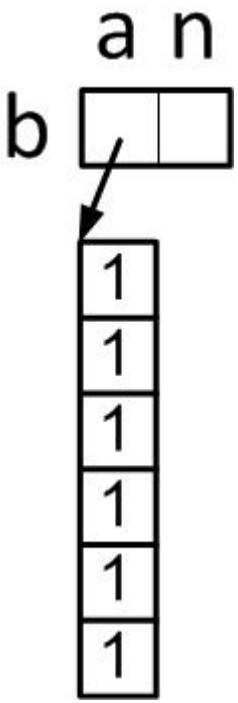
Сдвинуть строки матрицы вверх на одну циклически.

```
void main()
{ Matrix M(5,6,1); Array b;
  M.Show();
  b = M[0];
  for ( int i =1; i<5; i++)
    M[i-1] = M[i];
  M[4] = b;
  M.Show();
}
```

Задача




```
Matrix M(5,6,1); Array b;
b = M[0];
for ( int i =1; i<5; i++)
    M[i-1] = M[i];
M[4] = b;
```



Пример решения этой задачи со «сдвигом» *адресов* строк

```
int *& Array::Geta(){ return a;}

void Matrix::operator !() /* циклический сдвиг строк
                             вверх на одну позицию */
{
        int *b = a[0] . Geta();
        for(int  i = 1; i<m; i++)
            a[i - 1] . Geta() = a[i] . Geta();
         a[m - 1]. Geta() = b;
   }
```

class Array { int *a, n; ….};

```
 void Matrix:: operator <<( int k) // сдвиг на k
   {  while(k--)
      !(*this);
     }
void main()
{ Matrix M(5,6,1);
  cout<<M; // << перегружена для Matrix
  M<<2;      /* циклический сдвиг на 2
   позиции вверх */
 cout << endl<<M;
}
```

```
class A { int x, y;
          public:
          A(int xx = 0){ x = y = xx;
}...
```

```
class B{
// агрегированный класс
          A a;
          int z; ...
```

Разберемся, как в классе B можно использовать член-данные x и y из класса A

Определим функцию вывода в классе B:

void B :: Print() (1)

{ cout << ‘\n’ << **a.x** << ‘ ‘ << **a.y** << ‘ ‘ << z;}

Так нельзя: **a.x, a.y** - член-данные из части private класса A !

Способы выхода из положения

1. Использовать a.x и a.y ***непосредственно*** в член-функциях класса B станет возможным, если объявить ***класс B дружественным классу A***.

```
class B;
class A{ friend class B;
          ...                    };
```

```
void A :: Print()
{cout << '\n' << x << '  '<<y;}
```

2. Использовать **x** и **y** через член-функции класса A из части public:

```
void B :: Print() { a.Print(); cout << '
```

Конечно, нет.
a. это гарантирует!

Как вы думаете, не запустит ли компилятор тут рекурсию?

3. В некоторых случаях приходится специально задавать функции, которые возвращают то или иное значение из части private используемого класса.
Например,
в классе A - ч/функции **Getx(), Gety()**.

```
class A { int x, y;
          public:
          ...
          int Getx() { return x; }
          int Gety() { return y; } ...
```

Функция Print() в классе B в этом случае может быть определена таким образом

```
void B :: Print()
{ cout << ’\n’ << a.Getx() << ’ ‘ << a.Gety()
   << ’ ‘ << z;}
```

Пример использования агрегированных классов

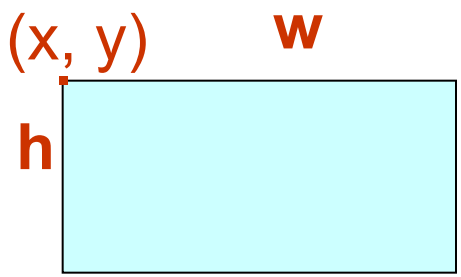
```
void main()
{ A a1, a2(7,8); // работают оба конструктора класса A
 B b1; /* работают конструкторы по умолчанию
    классов A и B: b1.a.x = b1.a.y = b1.z = 0; */
 B b2(1, 2, 3); /* работают конструкторы с
                   аргументом сначала A, потом B:
                      b2.a.x =1; b2.a.y = 2; b2.z = 3; */
B *b3= new B(7,8,9); // b3->a.x = 7, b3->a.y = 8, b3->z = 9
 a1. Print(); a2. Print(); // функция A :: Print()
 b1.Print(); b2.Print();   //  функция B:: Print()
b3->Print();                //                B::Print()
}
```

Графический пример агрегированного класса

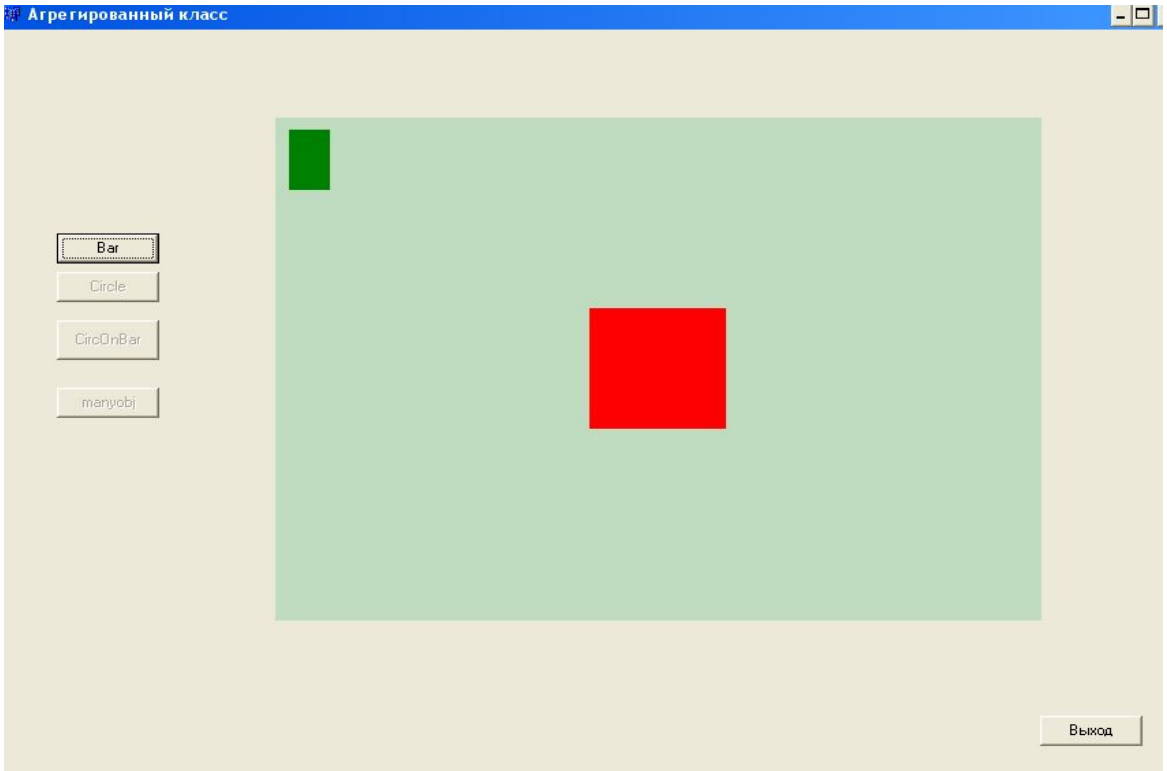
Определим 3 класса:

Bar – прямоугольник,
Circ – круг,
CircOnBar – круг на прямоугольнике

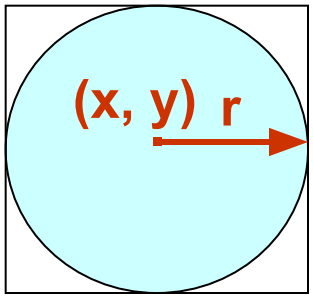
```
class Bar { int h, w, x, y;
        public:
      Bar(int hh, int ww, int xx, int yy)
       { h = hh; w = ww; x = xx; y = yy; }
      void Show(int c, TImage *im)
     { if(x+w>im->Width || y+h > im->Height || x<0 || y<0)
          throw “Выход за экран”;
       im->Canvas->Brush->Color = c;
       im-> Canvas-> FillRect( Rect(x, y, x+w, y+h));
       }
      };
```



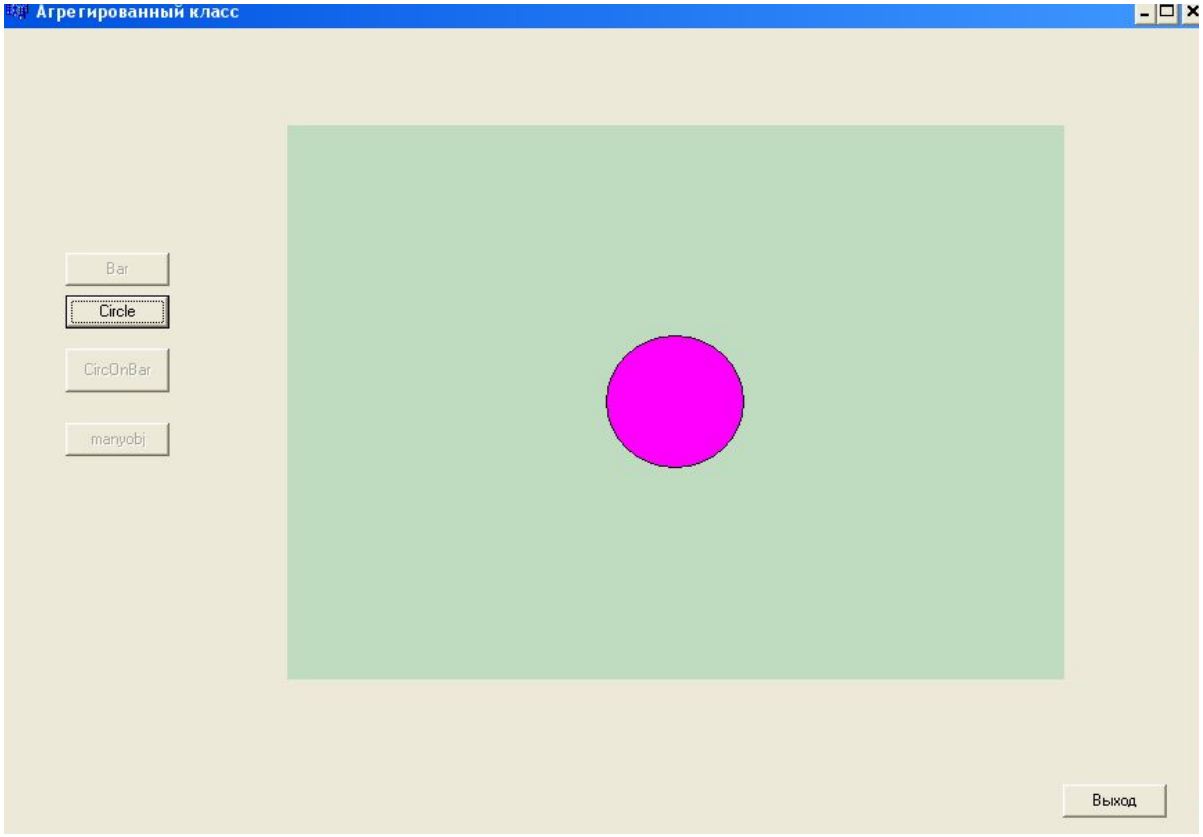
```
void __fastcall TForm1::Button1Click(TObject *Sender)
{ try{
  Bar b(50, 30, 10, 10);
  b.Show(clGreen, Image1);
  Bar c(100,100,Image1->Width/2-50,Image1->Height/2-50);
  c.Show(clRed, Image1) ;
  }catch(const char *s){ ShowMessage(s);}
   catch(...) { ShowMessage("Unknown error"); }
}
```



```
class Circ {
            int r, x, y;
            public:
           Circ(int rr, int xx, int yy)
             {r = rr; x = xx; y = yy;}
          void Show(int c, TImage *im)
 { if(x – r <0 || x+r>im->Width || y-r<0 || y+r>im->Height)
       throw «Выход за экран";
  im->Canvas->Brush->Color = c;
  im->Canvas->Ellipse( x-r, y-r, x+r, y+r);
  }
              };
```

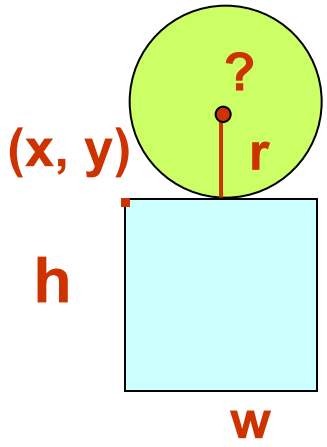


```
void __fastcall TForm1::Button3Click(TObject *Sender)
{ Circ c(50, Image1->Width/2, Image1->Height/2);
  c.Show(clFuchsia, Image1);
}
```

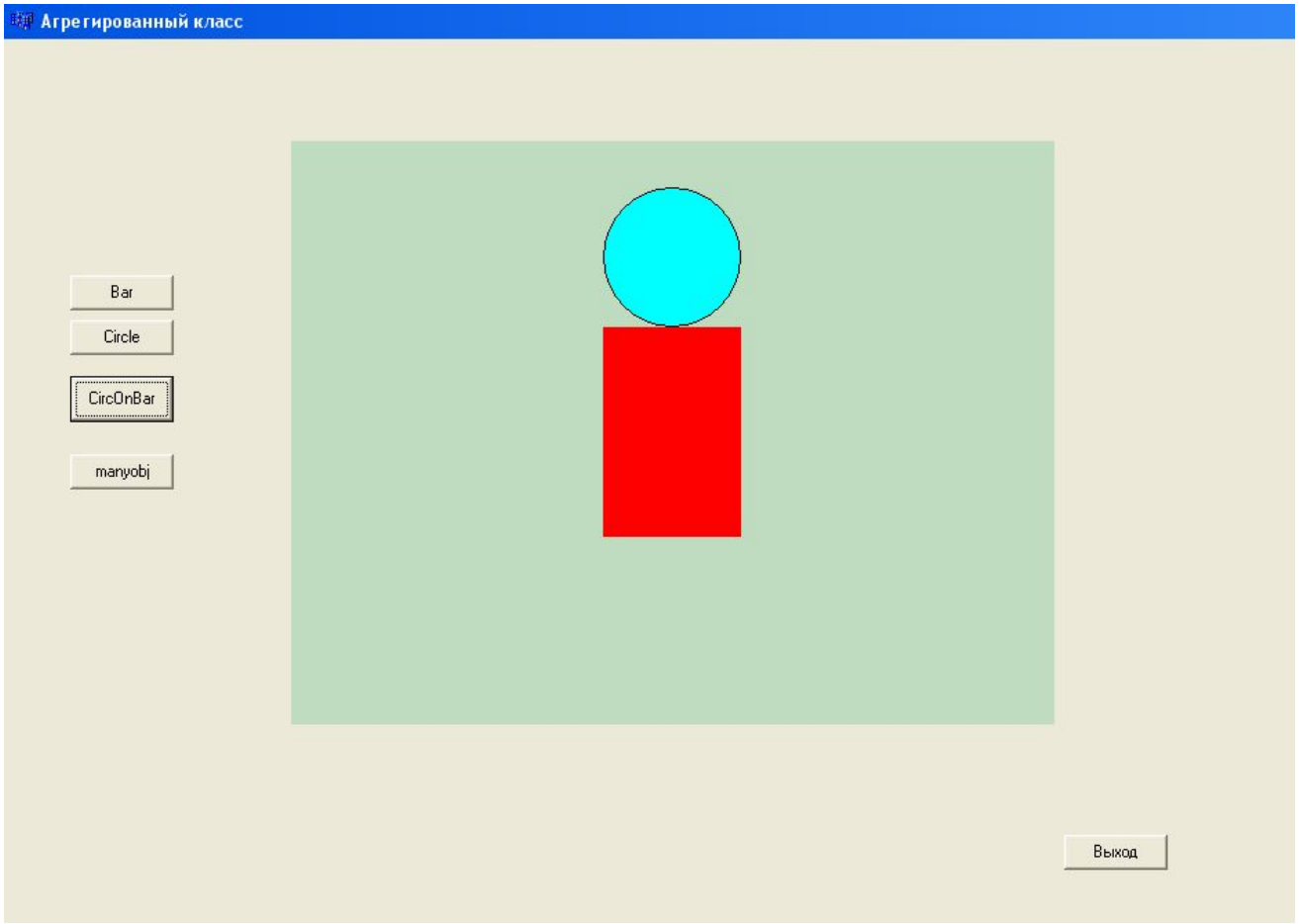


Агрегированный класс

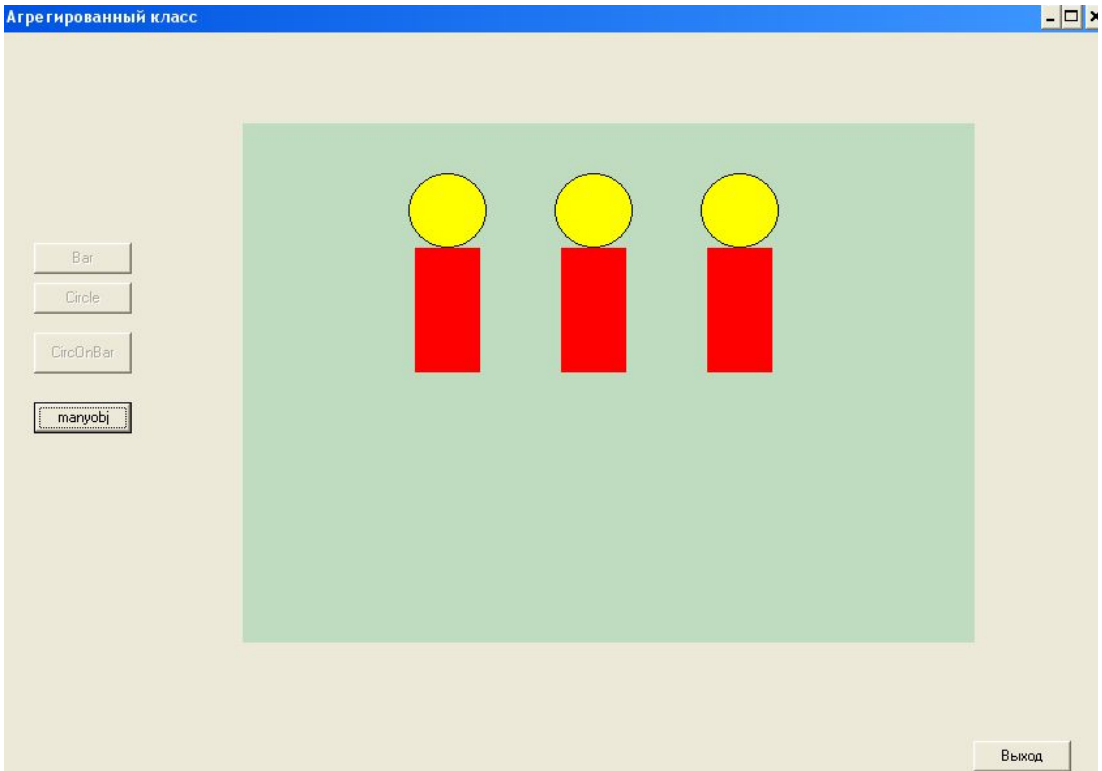
```
class CircOnBar{ Bar b;
                 Circ c;
                    public:
  CircOnBar(int h, int w, int x, int y, int r) : b(h, w, x, y),
           c(r, x + w/2, y - r)
   { }
 void Show(int cb, int cc, TImage *im)
 { b.Show(cb, im); c.Show(cc, im);
 }
                          };
```



```
void __fastcall TForm1::Button4Click(TObject *Sender)
{  CircOnBar cb(150, 100, Image1->Width/2-50, Image1->Height/2 - 75,
                    50);
   cb.Show(clRed, clAqua, Image1);
}
```



```
void __fastcall TForm1::Button5Click(TObject *Sender)
{
 int x = 20, dx = Image1->Width/5, i;
 //  Определяем массив агрегированных объектов CircOnBar
 CircOnBar bc[3] = {CircOnBar(100, 50, x = x + dx, 100, 30),
                    CircOnBar(100, 50, x = x + dx,100, 30),
                    CircOnBar(100, 50, x + dx, 100, 30)};
 for(i = 0; i<3; i++)                    h    w              y     r
 bc[i].Show(clRed, clYellow, Image1); // Выводим их на экран
```



```
// Объект в динамической области:
Image1->Canvas->Font->Size = 14;
Image1->Canvas->Font->Color = clRed;
Image1->Canvas->TextOut(420,360,«Объект ");
Image1->Canvas->TextOut(320,380,"в динамической
области!");
CircOnBar  *pbc = new CircOnBar(50, 50, 430, 300, 50);
pbc->Show(clRed, clPurple, Image1);
delete pbc; // удалили
}
```