Лекция 4

КОДТАРДЫ, ФУНКЦИЯ ШАБЛОНДАРЫН, ЕРЕКШЕЛІКТЕРДІ ӨҢДЕУДІ ҚАЙТА ҚОЛДАНУ МЕХАНИЗМІ

C++ тілінде ***мұрагерлік*** код пен абстракцияны қайта қолдануды жеңілдететін механизм болып табылады. Мұрагерлік программадағы кластар арасындағы байланысты орнатады. Мұрагерлікті қолдану арқылы жаңа кластар ескі кластар негізінде еншілес кластарға деректер элементтерін және мұралаушы кластағы функцияларды бірге қолдануға мүмкіндік бере отырып құрылуы мүмкін. Мұралаушы/еншілес қатынастарын белгілеу үшін қос нүкте (:) қолданылады.

```
class Employee { /* declaration of
parent */ };
class Manager : Employee { /*
declaration of child */ };
```

Жоғарыдағы листингте "is-a"-қатынасына енгізілген, яғни мұралау моделі "parent" және "child" терминдерін қолданатын кластармен жұмыс істеу үшін ***class Employee*** класы және ***class Manager*** класы хабарланады.

C++ тілінде еншілес класс деректердің барлық ***non-private*** элементтерін конструкторды қоса алғанда мұралайды.

Келесі листингте ата-ана класындағы әдістер қалай шақырылатынын көрсететін C++ тілінде мұрагерлік мысалы келтірілген. Бұл листингте ***SavingsAccount*** класы ***BankAccount*** класының мұрагер класы болып табылады. ***SavingsAccount*** класы ағымдағы тіркеу парақшасындағы ақшаға қол жеткізу тәсілін көрсетеді. 23-жол ата-ана класындағы конструкторды шақырады.

```
1  class BankAccount {
2
3  protected:
4      double sum;
5      string name;
6
7  public:
8
9      BankAccount(string nm) : name(nm), sum(0) {}
10
11     double balance() { return sum;}
12     void deposit(double amount) {sum += amount;}
13     void withdraw(double amount) {sum -= amount;}
14     string get_name() { return name;}
15 };
16
17 class SavingsAccount: public BankAccount {
18 protected:
19     double rate;
20
21 public:
22     SavingsAccount(string nm)
23             : BankAccount(nm),  // Call base class constructor
24     rate(0.055) {}
25
26     void add_interest() {sum *= (1 + rate);}
27     double get_rate() { return rate;}
28 };
```

Полиморфизм

Тізбектелген кластардың бір ерекшеліктері тізбектелген кластағы нұсқауыштар типі бойынша базалық кластағы нұсқауыштармен сәйкес келеді.

Кедесі листингте ***main*** функциясында ***CPolygon*** (ppoly1 және ppoly2) класы объектісіне нұсқайтын екі нұсқауыш құрылған. Содан соң ***rect*** және ***trgl*** сілтемелерін нұсқауыштарға меншіктейміз, себебі ол екеуі де ***CPolygon*** алынған еншілес кластар және екі меншіктеу де – орынды операциялар. ****ppoly1*** және ****ppoly2*** нұсқауыштарын ***rect*** және ***trgl*** объектілерімен бірге қолданудағы жалғыз шектеу ****ppoly1*** және ****ppoly2***

нұсқауыштары ***CPolygon**** типіне ие болады, сондықтан біз CPolygon-нан мұраланған ***CRectangle*** және ***CTriangle*** элементтеріне қайту үшін тек осы нұсқауыштарды ғана қолданамыз. Сондықтан біз ***area***() элемент- терін программа соңында шақырамыз, біз ***rect*** және ***trgl*** объектілерін ***ppoly1*** және ***ppoly2*** нұсқауыштарымен бірге қолдануымыз тиіс еді.

```
#include <iostream>
using namespace std;

class CPolygon {
protected:
    int width, height;
public:
    void set_values(int a, int b)  {
        width = a; height = b;
    }
};

class CRectangle : public CPolygon {
public:
    int area()  {
        return (width * height);
    }
};

class CTriangle : public CPolygon {
public:
    int area()  {
        return (width * height / 2);
    }
};

int main() {
    CRectangle rect;
    CTriangle trgl;
    CPolygon * ppoly1 = &rect;
    CPolygon * ppoly2 = &trgl;
    ppoly1->set_values(4, 5);
    ppoly2->set_values(4, 5);
    cout << rect.area() << endl;
    cout << trgl.area() << endl;

    return 0;
}
```

```
20
10
```

Виртуалды элементтер

Кластың туынды кластарында қайта қаралатын класс элементтерін ***виртуалды элементтер*** деп атайды. Класс элементін виртуалды элемент ретінде хабарлау үшін біз оны ***virtual*** кілттік сөзі арқылы хабарлауымыз керек:

```
#include <iostream>
using namespace std;

class CPolygon {
protected:
    int width, height;
public:
    void set_values(int a, int b)  {
        width = a; height = b;
    }
    virtual int area()  {
        return (0);
    }
};

class CRectangle : public CPolygon {
public:
    int area()  {
        return (width * height);
    }
};

class CTriangle : public CPolygon {
public:
    int area()  {
        return (width * height / 2);
    }
};

int main() {
    CRectangle rect;    CTriangle trgl;    CPolygon poly;
    CPolygon * ppoly1 = &rect;  CPolygon * ppoly2 = &trgl;  CPolygon * ppoly3 = &poly;
    ppoly1->set_values(4, 5);   ppoly2->set_values(4, 5);   ppoly3->set_values(4, 5);
    cout << ppoly1->area() << endl;
    cout << ppoly2->area() << endl;
    cout << ppoly3->area() << endl;

    return 0;
}
```

```
20
10
0
```

Енді үш класс (***CPolygon***, ***CRectangle*** және ***CTriangle***) бірдей элементтерден тұрады: ***width, height, set_values()*** және ***area***().

area() класының функция мүшесі базалық класта виртуалды болып хабарланады және ол әрбір туынды класта қайта қаралады. Егер, мысалы, ***CPolygon*** үшін ***area***() хабарламасында ***virtual*** кілттік сөзін өшірсек, содан соң программаны орындау үшін қайта іске қоссақ, келесі нәтижеге ие боламыз:

```
#include <iostream>
using namespace std;

class CPolygon {
protected:
    int width, height;
public:
    void set_values(int a, int b)  {
        width = a; height = b;
    }
/*virtual*/ int area()  {
            return (0);
    }
};

class CRectangle : public CPolygon {
public:
    int area()  {
        return (width * height);
    }
};

class CTriangle : public CPolygon {
public:
    int area()  {
        return (width * height / 2);
    }
};

int main() {
    CRectangle rect;    CTriangle trgl;     CPolygon poly;
    CPolygon * ppoly1 = &rect;  CPolygon * ppoly2 = &trgl;  CPolygon * ppoly3 = &poly;
    ppoly1->set_values(4, 5);   ppoly2->set_values(4, 5);   ppoly3->set_values(4, 5);
    cout << ppoly1->area() << endl;
    cout << ppoly2->area() << endl;
    cout << ppoly3->area() << endl;

    return 0;
}
```

Абстрактілі базалық кластар

Абстрактілі базалық кластар алдыңғы мысалдағы ***CPolygon*** класына ұқсас болып келеді. Негізгі ерекшелігі біз өткен мысалда ***CPolygon*** класында болатын ***area***() объектісіне арналған мүмкін болатын функцияны анықтадық, олай болса, абстрактілі базалық класта ***area***() функция мүшесін түрлендірмеуге мүмкіндігі бар. Тек ***area***() функция мүшесін нөлге теңестіру жеткілікті, мысалы:

```
class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area () =0;
};
```

Функцияның мұндай типі дәл виртуалды функцияны шақырады, кемінде бір нақты виртуалды функциядан тұратын барлық кластарды ***абстрактілі базалық класс*** деп атаймыз.

Регулярлы полиморфты кластар мен абстрактілі базалық кластар арасындағы негізгі айырмашылық абстрактілі базалық кластардың кемінде бір элементінде программаны өңдеу барысында қиындықтар туындайды. Бірақ объектілерге бағынбайтын класс толығымен қажетсіз емес, оларға нұсқауыштарды құрып, олардың полиморфты мүмкіндіктерін толығымен қолдануға болады.

Шаблондар

Функция шаблондары программистке функция логикасын қолдануға мүмкіндік береді. C++ тілінде функцияны хабарлау оның параметрлерінің деректер типімен анықталады. Мысалы, екі шаманың максимумын анықтау үшін біз функцияның барлық типтері үшін, яғни ***integers***, ***floats*** немесе ***strings*** типтері үшін есептеуді қайталауымыз керек. Шаблонды функциялар мүмкіндік программалаушыға мәліметтер типі мен параметрлерінен тәуелсіз функция құруға мүмкіндік береді.

```
int max(int x, int y) { return x < y
? y : x;}
float max(float x, float y) { return
x < y ? y : x;}
string max(string& x, string& y) {
return x < y ? y : x;}
```

Жоғарыдағы листинг функцияларды құрудың ең тиімсіз тәсілі болып келеді. Бірақ программаға функциялар шаблонын құру арқылы мәселені жеңілдетуге болады. Мысалы:

```
template <typename T>
T my_max(T x, T y) {
return x < y ? y
: x;   }
```

Листингтің бірінші жолында функцияның шаблонды екенін нұсқайтын ***template*** кілттік сөзі қолданылады. Бұрыштық жақшаға деректер типінің әмбебап есімі енгізіледі. Компилятор бұл есімді программада қолданылатын анықталған деректер типімен алмастырады. C++ программалау тілінде typename кілттік сөзінің орнына ***class*** кілттік сөзін қолдануға мүмкіндік береді.