

Katona József–Király Zoltán

OBJEKTUMORIENTÁLT SZOFTVERFEJLESZTÉS C++ NYELVEN

KONSTRUKTOR TÍPUSOK, DESTRUKTOR, OSZTÁLYSABLONOK,
FÜGGVÉNYSABLONOK ÉS FÜGGVÉNYTÚLTERHELÉSEK

© Katona József–Király Zoltán, 2015

Lektorálta: Dr. Kusper Gábor, tanszékvezető főiskolai docens

D▪U▪F PRESS
DUNAÚJVÁROSI FŐISKOLA
www.duf.hu

Kiadóvezető: Németh István
Felelős szerkesztő: Nemeskéry Artúr
Layout és tördelés: Duma Attila

Kiadja: DUF Press, a Dunaújvárosi Főiskola kiadója
Felelős kiadó: Dr. András István

Készült a HTSART nyomdában.
Felelős vezető: Halász Iván

ISBN 978-963-9915-55-8
ISSN 2415-9115

Katona József–Király Zoltán

OBJEKTUMORIENTÁLT SZOFTVERFEJLESZTÉS C++ NYELVEN

AZ OSZTÁLYOK BARÁTFÜGGVÉNYEI, OSZTÁLYOK
BARÁTSÁGA, ÖRÖKLÉS MECHANIZMUSA

Korai kötés – bezáródás jelenség

Késői kötések – virtuális tagfüggvények

Forráskódok



Dunakavics
D·U·F PRESS

Dunaújváros, 2015

Tartalom

1. Fejezet

Az osztályok barátfüggvényei	9
Osztályok barátsága	11
Gyakorló példák	12

2. Fejezet

Öröklés (Öröklődés) mechanizmusa	14
Korai kötés – bezáródás jelenség	16
Késői kötés – virtuális tagfüggvények	28
Öröklés több osztálytól	20
Gyakorló példák	22

3. Forráskódok

MuvBarat osztály deklarációja (Muvbarat.h)	27
MuvBarat osztály implementálása (MuvBarat.cpp)	28
MuvBarat osztály példányosítása (main.cpp)	29
Square osztály deklarációja (Square.h)	30
Rectangle osztály implementálása (Rectangle.cpp)	31
Square osztály példányosítása (main.cpp)	31
Muvelet osztály deklarációja (Muvelet.h)	32
UjMuvelet osztály deklarációja (UjMuvelet.h)	32

Muvelet osztály implementálása (Muvelet.cpp)	33
UjMuvelet osztály implementálása (UjMuvelet.cpp)	34
UjMuvelet osztály példányosítása (main.cpp)	35
Muvelet osztály deklarációja (Muvelet.h)	36
UjMuvelet osztály deklarációja (UjMuvelet.h)	36
Muvelet osztály implementálása (Muvelet.cpp)	37
UjMuvelet osztály implementálása (UjMuvelet.cpp)	38
UjMuvelet osztály példányosítása (main.cpp)	39
Adatok osztály deklarációja (Adatok.h)	40
Eredmeny osztály deklarációja (Eredmeny.h)	41
Muvelet osztály deklarációja (Muvelet.h)	41
MuveletiJel osztály deklarációja (MuveletiJel.h)	42
UjMuvelet osztály deklarációja (UjMuvelet.h)	42
Adatok osztály implementálása (Adatok.cpp)	43
Eredmeny osztály implementálása (Eredmeny.cpp)	43
Muvelet osztály implementálása (Muvelet.cpp)	44
MuveletiJel osztály implementálása (MuveletiJel.cpp)	45
UjMuvelet osztály implementálása (UjMuvelet.cpp)	46
UjMuvelet osztály példányosítása (main.cpp)	46
4. Felhasznált irodalom	48

Előszó

Úgy gondoljuk, hogy ez a könyvsorozat hozzájárulhat ahhoz is, hogy különböző szakemberek vagy akár középiskolai diákok is kamatoztathassák tudásukat a témában.

A műsorozat elsődleges célja az objektumorientált világ megismertetése. Az ismeretek elsajátításhoz igyekeztünk egy olyan magas szintű programozási nyelvet választani, amely az előismeretekre támaszkodva hatékonyan képes szemléltetni az OOP világában használt szabályokat, fogalmakat, elveket és modelleket. A választott C++ nyelv mindamelllett, hogy a fentebb leírt előírásoknak megfelel, a munkaerőpiacon is az egyik legsikeresebbnek számító magasszintű nyelv.

A szoftverfejlesztési és annak tanítási tapasztalataink alapján arra az elhatározásra jutotunk, hogy a választott C++ nyelvet nem a legelejétől kívánjuk ismertetni. Nem szeretnénk például elismételni azokat a vezérlési szerkezeteket, amelyeket már egy középszinten lévő programozó számtalanszor hallott és jó eséllyel használt. Természetesen ez nem azt jelenti, hogy ezeket az eszközöket el is felejthetjük, ugyanis, ahhoz, hogy jó programozó váljon belőlünk nem elég ismernünk ezeket az utasításkészleteket, hanem fel is kell ismernünk, hogy hol és mikor a legcélszerűbb azokat használni.

A könyv összeállításakor az ipari és oktatói tapasztalatainkra és számos, neves szakirodalomra támaszkodtunk. Reméljük, hogy ez a mű hozzásegíti az olvasót a tananyag megértéséhez és könnyed elsajátításához. Ehhez kellő kitartást és lelkesedést kívánunk.

Köszönettel tartozunk a kézirat lektorának, Dr. Kusper Gábornak gondos munkájáért, szakmai tanácsaiért és módszertani javaslataiért, melyek jelentősen hozzájárultak a könyvsorozat magasabb szakmai szintre emeléséhez.

A Szerzők

**OOP: angolul object-oriented programming egy programozási módszertan*

Objektumorientált programozás C++ nyelven

1. Fejezet

Az osztályok barátfüggvényei

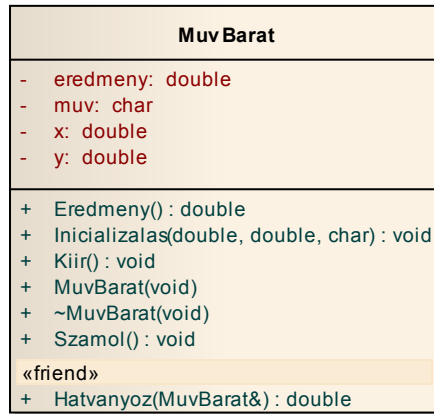
A **friend** (barát) mechanizmus lehetőséget biztosít az úgynevezett objektumorientáltság által biztosított védelem szelektív engedélyezésére. A mechanizmus lehetővé teszi, hogy az osztály **private** és **protected** tagjait nem csak az adott osztály tagfüggvényeivel manipulálhassuk. A **friend** deklarációt az osztály deklarációjában kell elhelyeznünk, paraméterként pedig az objektumpéldányt kell átadni. Tehát szükségünk lehet arra, hogy egy adott osztály magán adattagjait egy olyan függvénnyel is kezelhessük, amely nem eleme az adott osztálynak.

Tervezzünk a MuvBarat osztálynak egy barátfüggvényt, amely hatványt számol, azaz az x adattagját és az y adattagra emeli, a hatványozás eredményét pedig függvényértékként adja vissza! (CA_09)

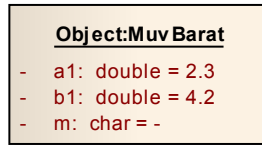
* Dunaiújvárosi Főiskola,
Informatika Intézet
E-mail: katonaj@mail.duf.hu

** Dunaiújvárosi Főiskola,
Informatika Intézet
E-mail: kiru@mail.duf.hu

Osztálydiagram (Class diagram)



Objektumdiagram (Object diagram)



A (*MuvBarat.h*) állományban találhatóak az osztály deklarációi. A deklaráció során a **friend** függvény is megfogalmazásra kerül. A barátfüggvény deklarációja:

friend double Hatvanyoz(const MuvBarat& v);

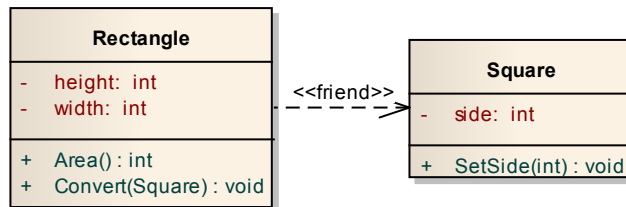
A függvény a paraméterben konstans referenciaként kapja meg a *MuvBarat* osztály példányát, hogy ne változtathassa meg az adattagokat, az eredményt pedig visszatérési értéként adja vissza. A (*MuvBarat.cpp*) állományban a **friend** függvény kidolgozása is látható. (Megjegyzem, hogy ekkor már nem kell kitenni a friend kulcsszót.) A (main.cpp) állomány a main() függvényben a Hatvanyoz() barátfüggvényt hívjuk.

Osztályok barátsága

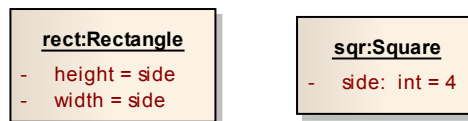
A C++ **friend** mechanizmusa nem merül ki csupán a függvényekkel, lehetőség van osztályok barátságát is kialakítani. Ha egy osztálynak létezik egy barátosztálya, akkor annak lehetősége van bármilyen láthatóságú adattag elérésére, legyen az akár **private** vagy **protected** jogú. Deklarálni abban az osztályban szükséges, amely felajánlja a jogait.

Írjunk egy olyan osztályt, ahol a négyzet osztálybarátja lesz a téglalap osztály. A négyzet osztály feladata az, hogy inicializálja az osztály adattagját, a téglalap osztály feladata pedig, hogy kiszámolja a síkidom területét, illetve át tudja alakítani a téglalapot négyzetté! (CA_10)

Osztálydiagram (Class diagram)



Objektumdiagram (Object diagram)

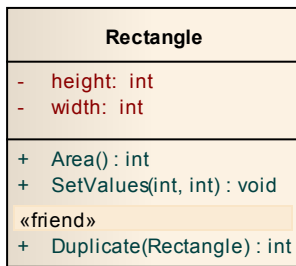


A (*Square.h*) állományban két barátosztály deklarációja látható. A négyzet osztálynak jelzésre kerül, hogy barátja a téglalap osztály, tehát a téglalap osztály a négyzet osztály minden adattagját eléri. (*Rectangle.cpp*) fájlban az osztályon kívül kidolgozásra kerül a *Convert()* függvény, ahol beállításra kerül a két téglalap oldalának a négyzet oldala. A főfüggvényben (*main.cpp*) létrehozásra kerül a két barátosztály objektuma. Tovább haladva a forráskódon, beállításra kerül a négyzet oldal (4 cm-re), majd meghívásra kerül a téglalap osztály *Convert()* tagfüggvénye, ahol paraméterül átadásra kerül a négyzet objektum és beállításra kerül a téglalap két oldala (4 cm-re), végül kiszámolásra kerül a téglalap síkidom területe a téglalap objektum segítségével.

Gyakorló példák

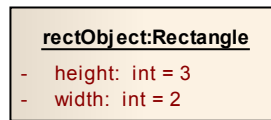
Írjon olyan friend-mechanizmust, ahol van egy barát- függvény, amelynek feladata az eredeti téglalap méreteinek a megduplázása. Az eredeti osztályban számolja ki a téglalap kerületét és területét, majd a paraméterben átadott objektum értékekkel dolgozva számolja ki a megkétszerezett téglalap kerületét és területét! Az osztály adattagjainak inicializálását most ne konstruktorral végezze! (GYF_CA_13)

Osztálydiagram (Class diagram)

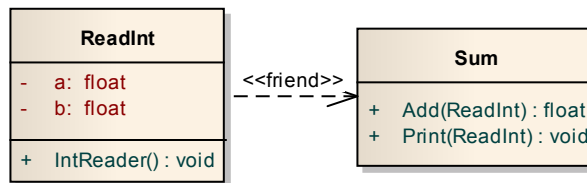


Írjon olyan friend-mechanizmust, ahol van két barát-osztály (ReadInt és Sum). Az első osztály legyen felelős két private jellegű valós típusú szám beolvasásáért. A második osztály az első osztály barátja legyen felelős a két valós szám összeadásáért és az eredmény kiírásáért. A számokat egészen addig olvassuk be, ameddig a felhasználó szeretné. (GYF_CA_14)

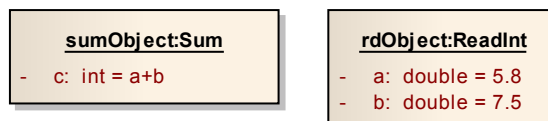
Objektumdiagram (Object diagram)



Osztálydiagram (Class diagram)



Objektumdiagram (Object diagram)



2. Fejezet

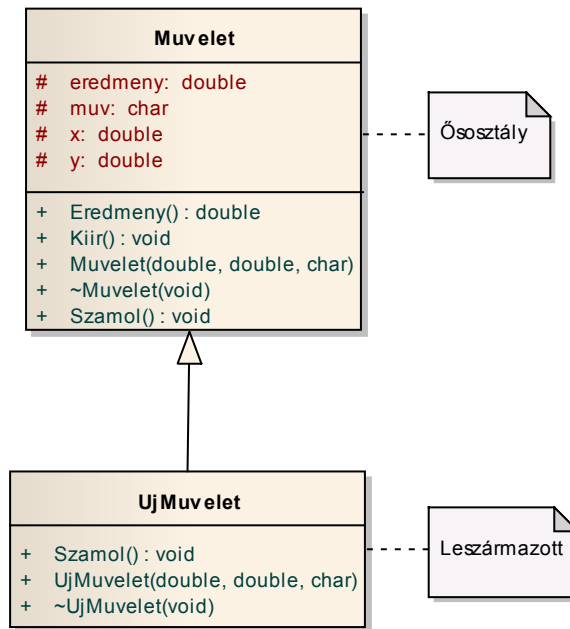
Öröklés (Öröklődés) mechanizmusa

Mi jut eszünkbe az öröklés szóról? Talán az, hogy a gyermek milyen tulajdonságokat örökölhet a szüleitől, vagy gondolhatunk egy családfára is. Ugyanilyen családfa építhető fel osztályokból is. Az objektum-orientáltság egyik legfőbb tulajdonsága az öröklés. Az öröklés jelentése, hogy egy már meglévő osztályból egy újabb osztályt hozunk létre (származtatunk). Az örököltetés során az új (kiterjesztett) osztály örököli a meglévő osztály (ős) nyilvános (**public**) és a védett (**protected**) adatait és viselkedését (függvények, indexelők, események és tulajdonságok), melyeket aztán sajátjaként felhasználhat. Az új osztály bővítheti is a meglévő osztályt, új adatakat, függvényeket stb. definiálhat, illetve újraértelmezheti az öröklött, de elavult viselkedési formákat (**polymorphism**). A származtatás szintaxisa a származtatandó osztály neve után kettőspont (:), majd a származtatási mód (**public**, **private**, **protected**) az osztály nevével. Többszörös öröklés esetén vesszővel elválasztva újabb osztályokat csatlakoztatunk a származtatási listához.

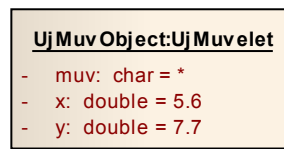
Az öröklés előnyei: Az öröklés segítségével újra felhasználhatóvá válnak a már megírt kódok és a már megtervezett adatstruktúrák. Ezáltal elkerülhető az adat és kód megkettőződése. A programokat könnyebben lehet karbantartani, mivel egy adott feladatot megvalósító kód és adatok általában a programban egyetlen, könnyen elérhető osztálydefinícióban szerepelnek ahelyett, hogy szétszórva helyezkednének el a forráskódban. Számos, valós világból származó kapcsolatot lehet hatékonyan modellezni a C++-ban definiált osztályhierarchiával.

Írjunk egy Muvelet osztályt, amelyből származtasunk egy UjMuvelet osztályt, amely a Szamol() tagfüggvényt újradefiniálja egy olyan változattal, amely a négy alapművelet mellett a hatványozás jelét (^) is kezeli. (CA_11)

Osztálydiagram (Class diagram)



Objektumdiagram (Object diagram)



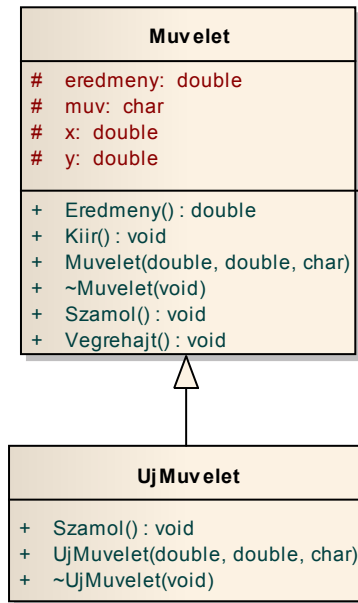
A (*Muvelet.h*) állomány tartalmazza a *Muvelet* osztály deklarációit. Az osztály adatait **protected** (védtett) láthatósággal láttuk el, azért, hogy a leszármaztatott osztályban is elérhetőek legyenek. Az állomány az adatok mellett tartalmazza az osztály-tagfüggvény prototípusait, amelyek majd a négy alapművelet elvégzésére lesznek képesek. Ez az osztályunk lesz az ősz vagy más néven szülő osztály, amelyet ki fogunk terjeszteni. Az (*UjMuvelet.h*) állomány tartalmazza a *Muvelet* osztály kiterjesztésének a deklarációit.

A konstruktoron és a destruktoron kívül az osztály tartalmaz még egy `Szamol()` függvényt, amely a négy alpműveletet végrehajtásán kívül képes lesz a hatványozásra is. A (*Muvelet.cpp*) fájl a *Muvelet* osztály tagfüggvényeinek kidolgozását fogja tartalmazni. Láthatjuk, hogy az ősoosztályban a `Szamol()` tagfüggvény még csak a négy alpműveletre képes. Az (*UjMuvelet.cpp*) állomány fogja tartalmazni a *Muvelet* osztály kiterjesztésének tagfüggvényeinek a kidolgozását. Fontos megjegyezni, hogy először mindig az ősoosztály konstruktorát látjuk el kezdő értékkel, majd a leszármaztatott osztály adattagjait inicializáljuk. Ebben az osztályban a `Szamol()` függvény már képes a hatványozásra is. A (*main.cpp*) állományban két számon fogjuk elvégezni a lehetséges matematikai műveleteket. Ahogy látjuk csak a kiterjesztett osztályt példányosítjuk, hiszen minden olyan tulajdonságra képes, amire a szülő osztály.

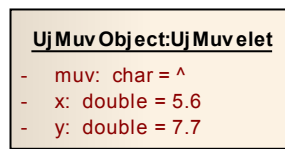
Korai kötés – bezáródás jelenség

Legyen a *Muvelet* osztály új tagfüggvénye a *Vegrehajt()*, amely a saját *Szamol()* tagfüggvényt aktiválja! Mint láttuk a *Muvelet* osztályból származtatott *UjMuvelet* osztálynak is van egy *Szamol()* tagfüggvénye, amely a négy alpművelet mellett kezeli a hatványozás műveletét (^) Az *UjMuvelet* objektumpéldánya aktiválja az öröklött *Vegrehajt()* tagfüggvényt! Mit tapasztalunk? (CA_12)

Osztálydiagram (Class diagram)



Objektumdiagram (Object diagram)



A feladattal kapcsolatban a következő deklarációkat, implementálásokat és példányosításokat találjuk a mellékletben:

- *Muvelet osztály deklarációja (Muvelet.h)* deklarációja.
- *UjMuvelet osztály deklarációja (UjMuvelet.h)* deklarációja.
- *Muvelet osztály implementálása (Muvelet.cpp)* állomány tartalma.
- *UjMuvelet osztály implementálása (UjMuvelet.cpp)* állomány tartalma.
- *A UjMuvelet osztály példányosítása (main.cpp)* forráskódja.

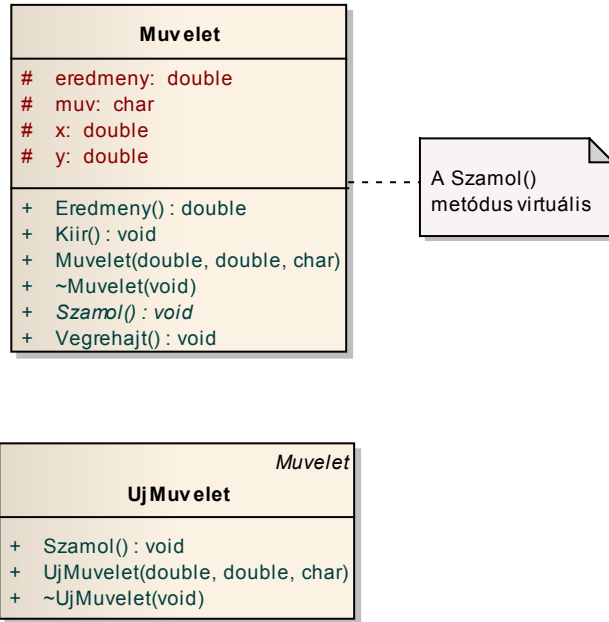
A program futtatását követően azt tapasztaljuk, hogy a $\wedge$ műveleti jelet érvénytelennek jelzi a program, mivel a *Vegrehajt()* a *Muvelet* osztály *Szamol()* tagfüggvényét hajtotta végre. Ez a jelenség a korai kötés (**early binding**), mert már a fordítási fázisban eldől, hogy az alaposztály *Szamol* nevű függvény hívódik meg.

Késői kötés – virtuális tagfüggvények

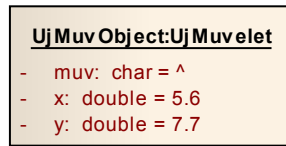
Az előző feladatban keletkezett problémát a virtuális tagfüggvények használatával orvosolhatjuk. Az alaposztály virtuális tagfüggvénye teljesen újradefiniálható (lecserélhető) a származtatott osztály azonos nevű és paraméterezésű tagfüggvényeivel. Ezzel a származtatási hierarchia valamely szinten, mindig az éppen ott definiált függvényváltozata lesz érvényben. Így megvalósul az objektumok sokalakúsága (a **polymorphism**).

Az előző feladatban annyi módosítást végezzünk el, hogy a *Muvelet* osztályban a *Szamol()* tagfüggvényt tegyük virtuálissá! Az *UjMuvelet* változatlanul aktiválja a *Vegrehajt()* tagfüggvényt! Mit tapasztalunk? (CA_13)

Osztálydiagram (Class diagram)



Objektumdiagram (Object diagram)



A feladattal kapcsolatban a következő deklarációkat, implementálásokat és példányosításokat találjuk a mellékletben:

- *Muvelet osztály deklarációja (Muvelet.h)* deklarációja.
- *UjMuvelet osztály deklarációja (UjMuvelet.h)* deklarációja.
- *Muvelet osztály implementálása (Muvelet.cpp)* állomány tartalma.
- *UjMuvelet osztály implementálása (UjMuvelet.cpp)* állomány tartalma.
- *Az UjMuvelet osztály példányosítása (main.cpp)* forráskódja.

Azt tapasztaltuk, hogy jó lesz az eredmény, amely azt jelenti, hogy a *Vegrehajt()* tagfüggvény függvényében az *UjMuvelet* osztály *Szamol()* tagfüggvénye hívódott meg. Ez az úgynevezett késői kötés, amikor futás során derül ki, hogy melyik tagfüggvényt kell aktiválni. A **virtual** teszi lehetővé a megfelelő tagfüggvények hívását, azaz a késő kötés (**late binding**) megvalósítását.

Öröklés több osztálytól

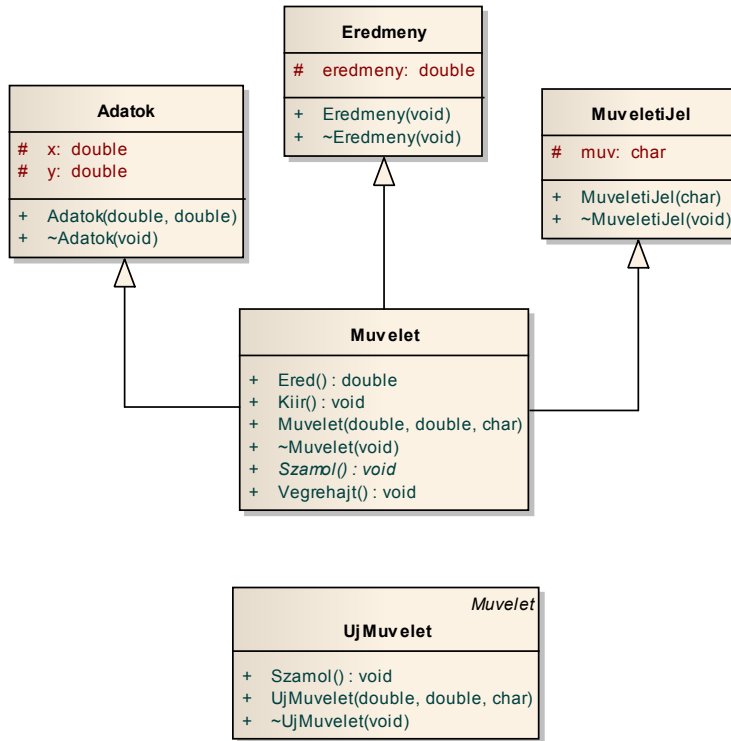
Az öröklődés mechanizmusa lehetővé teszi számunkra azt, hogy egy kiterjesztett osztálynak ne csak egy őse legyen, hanem több is. Ezzel a tulajdonsággal azonban óvatosan kell bánni, mert könnyen képes a kódot átláthatatlanná tenni. Az ősosztályok felsorolását a (,) karakterrel végezzük.

Oldjuk meg a mintafeladatot úgy, hogy az adattagok tárolására hozzunk létre önálló osztályokat, és a Muvelet osztályt azoktól származtassuk! Az UjMuvelet osztály a Muvelet osztálytól származtatva változatlanul úgy működjön, mint az előző feladatban láttuk! (CA_14)

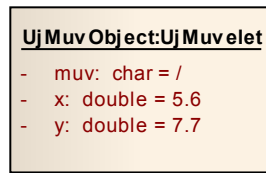
Három osztályra osztottuk szét az adattagokat:

- Adatok osztály: adattagjai az *x* és *y* operandusok. A konstruktor az adattagoknak ad kezdőértéket;
- MuveletiJel osztály: adattagja a *muv* műveleti jelet tárolja a művelethez. Konstruktora a műveleti jelenek ad kezdőértéket;
- Eredmeny osztály: adattagja a művelet eredményét tároló *eredmeny*. Konstruktora zérus kezdőértéket ad az eredmény adatagnak.

Osztálydiagram (Class diagram)



Objektumdiagram (Object diagram)



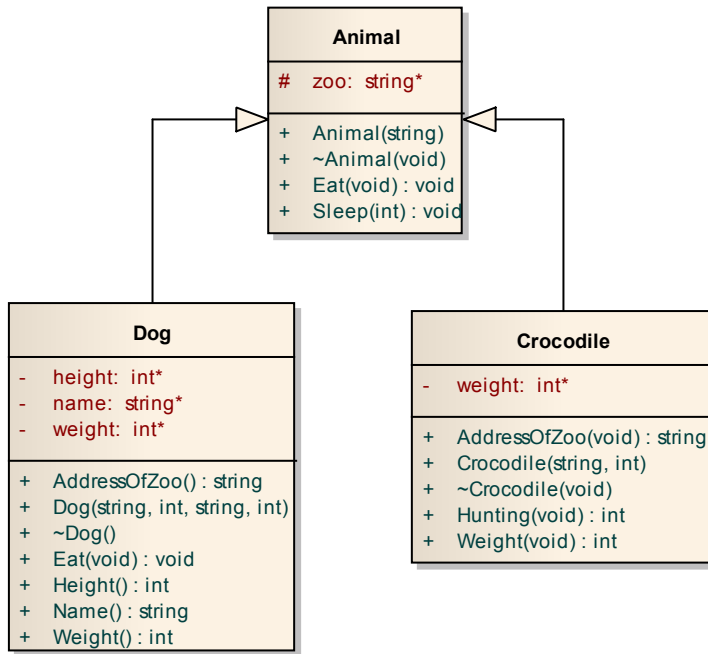
A feladattal kapcsolatban a következő deklarációkat, implementálásokat és példányosításokat találjuk a mellékletben:

- *Adatok osztály deklarációja (Adatok.h) deklarációja.*
- *Eredmeny osztály deklarációja (Eredmeny.h) deklarációja.*
- *MuveletiJel osztály deklarációja (MuveletiJel.h) deklarációja.*
- *Muvelet osztály deklarációja (Muvelet.h) deklarációja.*
- *UjMuvelet osztály deklarációja (UjMuvelet.h) deklarációja.*
- *Adatok osztály implementálása (Adatok.cpp) állomány tartalma.*
- *Muvelet osztály implementálása (Muvelet.cpp) állomány tartalma.*
- *MuveletiJel osztály implementálása (MuveletiJel.cpp) állomány tartalma.*
- *UjMuvelet osztály implementálása (UjMuvelet.cpp) állomány tartalma.*
- *UjMuvelet osztály példányosítása (main.cpp) forráskódja.*

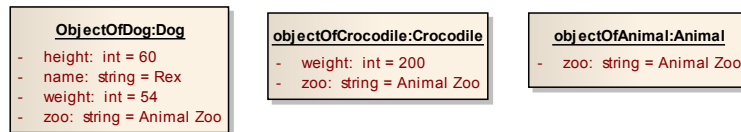
Gyakorló példák

Írjunk egy programot, amelynek van egy Animal osztálya. Az osztály legyen képes az állatokat etetni és altatni. A Dog osztály az Animal osztályból származzon le és legyen képes az önálló evésre, amely során a súlya 1 kg-ot gyarapszik, ez a függvény az ősoosztály felüldefiniált függvénye legyen. Továbbá a Dog osztály legyen képes megmondani az állat lakhelyét. A kutyának legyen egy neve, marmagassága és testsúlya. Az Animal osztályból öröklődjön még a Crocodile osztály, amely képes vadászni és minden vadászat során gyarapszik, a súlya, 5 kg-ot. Írjunk egy egyszerű main függvényt, ahol kiírjuk az állatok állapotát. (GYF_CA_15)

Osztálydiagram (Class diagram)

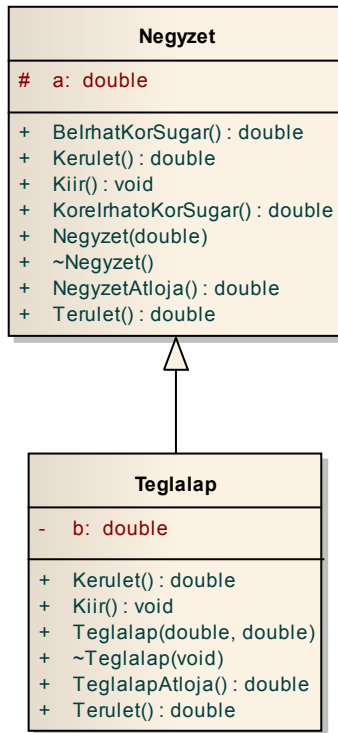


Objektumdiagram (Object diagram)



Írjunk egy programot, amelynek van egy Negyzet osztálya. Az osztály legyen képes a négyzet átlóját, területét, kerületét, a bele írható kör sugarát és a köré írható kör sugarát kiszámolni. Továbbá ezeket az értékeket írjuk ki a konzolra. A Teglalap osztály a Negyzet osztályból származzon le és legyen képes a téglalap átlóját, területét és kerületét kiszámolni, majd az eredményt kiírni a konzolra. A téglalap oldalait úgy adjuk meg, hogy az egyik oldal hossza az a négyzet oldal hossza is legyen egyben. Írjunk egy egyszerű main függvényt, ahol kiírjuk az adatokat. (GYF_CA_16)

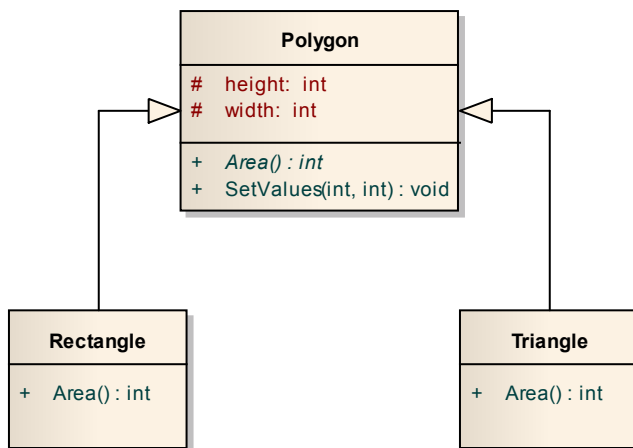
Osztálydiagram (Class diagram)



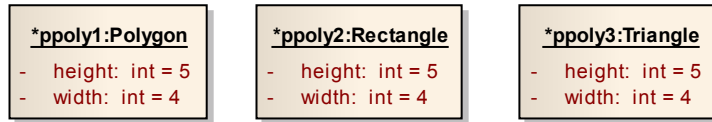
Objektumdiagram (Object diagram)



Írjunk egy programot, amelynek van egy Polygon osztálya. Az osztálynak legyen két protected láthatóságú adattagja, továbbá legyen két tagfüggvénye, az egyikben inicializáljuk az osztály két adattagját, a másikban területet számolunk, virtuális függvény segítségével és a visszaérési értéke egy egész szám, az ősoosztályban a függvény adjon vissza nullát. A Rectangle osztály a Polygon osztály kiterjesztése legyen, ahol az ősoosztály virtuális függvényét felhasználva számoljuk ki a téglalap területét. Az osztálynak nincs adattagja, az ősoosztálybeli adattagokat használjuk a terület számolásához. A Triangle osztály hasonlóan a téglalap osztályhoz hasonlóan számolja ki a háromszög területét. (GYF_CA_17)

Osztálydiagram (Class diagram)

Objektumdiagram (Object diagram)



3. Forráskódok

MUVBARAT OSZTÁLY DEKLARÁCIÓJA (MUVBARAT.H)

```

/*****
* MuvBarat osztály deklarációja
* Demonstrációs célok:
*         -> friend (barát) függvények
*         -> konstans referencia
*
* Katona József      <katonaj@mail.duf.hu>
*****/
#ifndef MUVBARAT_H //if not define egy szimbólum nemlétét ellenőrzi
#define MUVBARAT_H //egy szimbólum definiálása és értékének megadása
//=====
class MuvBarat {
//=====
private:
    double x, y, eredmeny;
    char muv;

public:
    MuvBarat::MuvBarat(void);
    void Inicializalas(double x1, double y1, char muv1);
    void Szamol();
    double Eredmeny() { return eredmeny; }
    void Kiir();
    //barát függvény deklarálása
    friend double Hatvanyoz(const MuvBarat& v);
    MuvBarat::~MuvBarat(void);
};
#endif //feltétel lezárása
```

MUVBARAT OSZTÁLY IMPLEMENTÁLÁSA (MUVBARAT.CPP)

```

/*****
* MuvBarat osztály definiálása/implementálása
* Demonstrációs célok:
*     -> friend (barát) függvények kifejtése
*     -> objektum paraméter
*
* Katona József      <katonajozsef@mail.duf.hu>
*****/
#include <cmath>           //általános matematikai műveletek
#include <iostream>        //C++ alapvető adatfolyam I/O rutinok
#include "MuvBarat.h"      //a MuvBarat osztály deklarációi

using namespace std;      //az std névtér behívása

//-----
MuvBarat :: MuvBarat(void) {}
//-----

//-----
void MuvBarat :: Inicializalas(double x1, double y1, char muv1)
//-----
{
    x = x1; y = y1; muv = muv1;
}

//-----
void MuvBarat :: Szamol()
//-----
{
    switch(muv)
    {
        case '+': eredmeny = x + y; break;
        case '-': eredmeny = x - y; break;
        case '*':  eredmeny = x * y; break;
        case '/':  eredmeny = x / y; break;
    }
}

//-----
void MuvBarat :: Kiir()
//-----
{
    cout << x << " " << muv << " " << y
         << " = " << eredmeny << endl;
}

```

```
//-----
double Hatvanyoz(const MuvBarat& v)           //friend függvény
//-----
{
    return pow(v.x, v.y);
}

//-----
MuvBarat::~MuvBarat(void) {}
//-----
```

MUVBARAT OSZTÁLY PÉLDÁNYOSÍTÁSA (MAIN.CPP)

```
/******
 * MuvBarat osztály példányosítása
 * Demonstrációs célok:
 *     -> MuvBarat osztály példányosítása
 *     -> Friend függvény hívása
 *
 * Katona József    <katonaj@mail.duf.hu>
 *****/
#include <iostream>           //C++ alapvető adatfolyam I/O rutinok
#include <conio.h>             //DOS konzol I/O rutinok hívása
#include "MuvBarat.h"        //a MuvBarat osztály deklarációi

using namespace std;         //az std névtér behívása

/******
int main()
//*****
{
    double a1 = 2.3, b1 = 4.2;
    char m = '-';

    MuvBarat objektum;
    objektum.Inicializalas(a1, b1, m);
    objektum.Szamol();
    objektum.Kiir();
    cout << "Hatvany: " << Hatvanyoz(objektum) << endl; //barátfüg

    _getch();
    return 0;
}
```

SQUARE OSZTÁLY DEKLARÁCIÓJA (SQUARE.H)

```

/*****
 * Square osztály deklarációja
 * Demonstrációs célok:
 *      -> friend mechanizmus (osztályok barátsága)
 *
 * Katona József      <katonaj@mail.duf.hu>
 *****/
#ifndef SQUARE_H //if not define egy szimbólum nemlétét ellenőrzi
#define SQUARE_H //egy szimbólum definiálása és értékének megadása
//=====
class Square {
//=====
private:
    int side;
public:
    void SetSide (int a) { side = a; }

    friend class Rectangle; //Az négyzet osztály barátja a
                             //téglalap osztály
};
#endif //feltétel lezárása

#ifndef RECTANGLE_H //if not define egy szimbólum nemlétét ellenőrzi
#define RECTANGLE_H //egy szimbólum definiálása és értékének megadása
//=====
class Rectangle {
//=====
private:
    int width, height;
public:
    int Area() {return (width * height);}
    void Convert (Square a);
    /*{
        width = a.side; //elérhető a négyzet osztály private
        height = a.side; //adattagja is
    }*/
};
#endif //feltétel lezárása

```

RECTANGE OSZTÁLY IMPLEMENTÁLÁSA (RECTANGLE.CPP)

```

/*****
 * Rectangle osztály definiálása/implementálása
 * Demonstrációs célok:
 *      -> friend mechanizmus
 *
 * Katona József      <katonaj@mail.duf.hu>
 *****/
#include "Square.h"           //a Square osztály deklarációi

//-----
void Rectangle :: Convert(Square a)
//-----
{
    width = a.side; //elérhető a négyzet osztály private
    height = a.side; //adattagja is
}

```

SQUARE OSZTÁLY PÉLDÁNYOSÍTÁSA (MAIN.CPP)

```

/*****
 * Square osztály példányosítása
 * Demonstrációs célok:
 *      -> friend mechanizmus
 *
 * Katona József      <katonaj@mail.duf.hu>
 *****/
#include <iostream>           //C++ alapvető adatfolyam I/O rutinok
#include <conio.h>             //DOS konzol I/O rutinok hívása
#include "Square.h"          //a Square osztály deklarációi

using namespace std;         //az std névtér behívása

//*****
int main ()
//*****
{
    Square sqr;
    Rectangle rect;
    sqr.SetSide(4);
    rect.Convert(sqr);
    cout << rect.Area();

    _getch();
    return 0;
}

```

MUVELET OSZTÁLY DEKLARÁCIÓJA (MUVELET.H)

```

/*****
* Muvelet osztály deklarációja
* Demonstrációs célok:
*     -> öröklődés mechanizmusa
*     -> kód újra felhasználás
*     -> őssztály
*     -> osztályok kiterjesztése
*
* Katona József      <katonaj@mail.duf.hu>
*****/
#ifndef MUVELET_H //if not define egy szimbólum nemlétét ellenőrzi
#define MUVELET_H //egy szimbólum definiálása és értékének megadása
//=====
class Muvelet {
//=====
    protected: //öröklés esetén szerencsésebb a protected elérés használata
        double x, y, eredmény;
        char muv;

    public:
        Muvelet(double x1, double y1, char muv1);
        ~Muvelet(void);
        void Szamol();
        double Eredmeny() {return eredmény;}
        void Kiir();
};
#endif //feltétel lezárása

```

UJMUVELET OSZTÁLY DEKLARÁCIÓJA (UJMUVELET.H)

```

/*****
* UjMuvelet osztály deklarációja
* Demonstrációs célok:
*     -> öröklődés mechanizmusa
*     -> kód újra felhasználás
*     -> őssztály
*     -> osztályok kiterjesztése
*
* Katona József      <katonaj@mail.duf.hu>
*****/
#ifndef UJMUVELET_H //if not define egy szimbólum nemlétét ellenőrzi
#define UJMUVELET_H //egy szimbólum definiálása és értékének megadása
#include "Muvelet.h"//A Muvelet osztály deklarációi

```

```
//=====
class UjMuvelet : public Muvelet { //leszármaztatás
//=====
public:
    UjMuvelet(double x1, double x2, char muv1);
    ~UjMuvelet(void);
    void Szamol();
};
#endif //feltétel lezárása
```

MUVELET OSZTÁLY IMPLEMENTÁLÁSA (MUVELET.CPP)

```
/******
* Muvelet osztály definiálása/implementálása *
* Demonstrációs célok: *
* -> öröklődés mechanizmusa *
* -> kód újra felhasználás *
* -> őszosztály *
* -> osztályok kiterjesztése *
* *
* Katona József <katonaj@mail.duf.hu> *
*****/
#include <iostream> //C++ alapvető adatfolyam I/O rutinok
#include "Muvelet.h" //a Muvelet osztály deklarációi
using namespace std; //az std névtér behívása

//-----
Muvelet::Muvelet(double x1, double y1, char muv1)
//-----
{
    x = x1; y = y1; muv = muv1; eredmeny = 0;
}

//-----
Muvelet::~Muvelet(void) {}
//-----

//-----
void Muvelet::Szamol()
//-----
{
    . . . . .
}
```

```

switch(muv)
{
    case '+': eredmeny = x + y; break;
    case '-': eredmeny = x - y; break;
    case '*': eredmeny = x * y; break;
    case '/': eredmeny = x / y; break;
}

//-----
void Muvelet::Kiir()
//-----
{
    cout << x << " " << muv << " " << y << " = " << eredmeny << endl;
}

```

UjMUVELET OSZTÁLY IMPLEMENTÁLÁSA (UjMUVELET.CPP)

```

/*****
* UjMuvelet osztály definiálása/implementálása
* Demonstrációs célok:
*     -> öröklődés mechanizmusa
*     -> kód újra felhasználás
*     -> őszosztály
*     -> osztályok kiterjesztése
*
* Katona József <katonaj@mail.duf.hu>
*****/
#include <cmath> //matematikai függvények
#include "UjMuvelet.h" //az UjMuvelet osztály deklarációi

//-----
UjMuvelet::UjMuvelet(double x1, double y1, char muv1): Muvelet(x1,y1,muv1)
//-----
    // paraméterek átadva a Muvelet konstruktorának
{} //üres a törzse, muvel nincs saját inicializálható adattagja

//-----
UjMuvelet::~UjMuvelet(void) {}
//-----

//-----
void UjMuvelet::Szamol()
//-----
{

```

```

        if (muv == '^')
            eredmeny = pow(x,y); //ez a bővített művelet
            //a pow() igényli a cmatch fejlécállományt
        else
            Muvelet::Szamol();
    }

```

UjMUVELET OSZTÁLY PÉLDÁNYOSÍTÁSA (MAIN.CPP)

```

/*****
* UjMuvelet osztály példányosítása
* Demonstrációs célok:
* -> kiterjesztett osztály példányosítása
*
* Katona József <katonaj@mail.duf.hu>
*****/
#include <iostream> //C++ alapvető adatfolyam I/O rutinok
#include <cstring> //C++ verziója, C "string.h"-nak
#include <conio.h> //DOS konzol I/O rutinok hívása
#include "UjMuvelet.h" //az UjMuvelet osztály deklarációi

using namespace std; //az std névtér behívása

/*****
int main()
/*****
{
    double a1, b1;
    char m;

    cout << "1. adat: "; cin >> a1;
    cout << "2. adat: "; cin >> b1;

    do
    {
        cout << "muveleti jel (+,-,*,/,^): "; cin >> m;
    } while(!strchr("+-*/^",m));

    UjMuvelet UjMuvObject(a1,b1,m);
    UjMuvObject.Szamol();
    UjMuvObject.Kiir();

    _getch();
    return 0;
}

```

MUVELET OSZTÁLY DEKLARÁCIÓJA (MUVELET.H)

```

/*****
* Muvelet osztály deklarációja
* Demonstrációs célok:
*     -> öröklődés mechanizmusa
*     -> kód újra felhasználás
*     -> őssztály
*     -> osztályok kiterjesztése
*     -> koria kötés problémája
*     -> inline tagfüggvény
*
* Katona József    <katona.jozsef.duf@freemail.hu>
*****/
#ifndef MUVELET_H //if not define egy szimbólum nemlétét ellenőrzi
#define MUVELET_H //egy szimbólum definiálása és értékének megadása
//=====
class Muvelet {
//=====
protected:
    double x, y, eredmeny;
    char muv;

public:
    Muvelet(double x1, double y1, char muv1);
    ~Muvelet(void);
    void Szamol();
    double Eredmeny() {return eredmeny;}
    void Kiir();
    void Vegrehajt() { Szamol(); } //új inline tagfüggvény
};
#endif //feltétel lezárása

```

UjMUVELET OSZTÁLY DEKLARÁCIÓJA (UjMUVELET.H)

```

/*****
* UjMuvelet osztály deklarációja
* Demonstrációs célok:
*     -> öröklődés mechanizmusa
*     -> kód újra felhasználás
*     -> őssztály
*     -> osztályok kiterjesztése
*     -> koria kötés problémája
*     -> inline tagfüggvény
*

```

```

* Katona József      <katonaj@mail.duf.hu>
*****/
#ifndef UJMUVELET_H //if not define egy szimbólum nemlétét ellenőrzi
#define UJMUVELET_H //egy szimbólum definiálása és értékének megadása
#include "Muvelet.h"//A Muvelet osztály deklarációi

//=====
class UjMuvelet : public Muvelet {
//=====
public:
    UjMuvelet(double x1, double x2, char mov1);
    ~UjMuvelet(void);
    void Szamol();
};

#endif //feltétel lezárása

```

MUVELET OSZTÁLY IMPLEMENTÁLÁSA (MUVELET.CPP)

```

/*****
* Muvelet osztály implementálása
* Demonstrációs célok:
*      -> öröklődés mechanizmusa
*      -> kód újra felhasználás
*      -> ósosztály
*      -> osztályok kiterjesztése
*      -> korai.kötés problémája
*      -> inline tagfüggvény
*
* Katona József      <katonaj@mail.duf.hu>
*****/
#include <iostream>      //C++ alapvető adatfolyam I/O rutinok
#include "Muvelet.h"     //a Muvelet osztály deklarációi

using namespace std;     //az std névtér behívása

//-----
Muvelet::Muvelet(double x1, double y1, char mov1)
//-----
{
    x = x1; y = y1; mov = mov1; eredmeny = 0;
}

```

```
//-----
Muvelet::~Muvelet(void)
//-----
{
}

//-----
void Muvelet::Szamol()
//-----
{
    switch(muv)
    {
        case '+': eredmény = x + y; break;
        case '-': eredmény = x - y; break;
        case '*': eredmény = x * y; break;
        case '/': eredmény = x / y; break;
        default:
            eredmény = 0;
            cerr << "Ervenytelen muveleti jel: " << muv << endl;
    }
}

//-----
void Muvelet::Kiir()
//-----
{
    cout << x << " " << muv << " " << y << " = " << eredmény << endl;
}
```

UjMUVELET OSZTÁLY IMPLEMENTÁLÁSA (UjMUVELET.CPP)

```
/******
* UjMuvelet osztály implementációja
* Demonstrációs célok:
*     -> öröklődés mechanizmusa
*     -> kód újra felhasználás
*     -> őssztály
*     -> osztályok kiterjesztése
*     -> korai kötés problémája
*     -> inline tagfüggvény
*
* Katona József <katona.jozsef.duf@freemail.hu>
*****/
#include <cmath> //általános matematikai műveletek
#include "UjMuvelet.h" //az UjMuvelet osztály deklarációi
```

```
//-----
UjMuvelet::UjMuvelet(double x1, double y1, char muv1): Muvelet(x1,y1,muv1) {}
//-----

//-----
UjMuvelet::~UjMuvelet(void) {}
//-----

//-----
void UjMuvelet::Szamol()
//-----
{
    if (muv == '^')
        eredmeny = pow(x,y);
    else
        Muvelet::Szamol();
}
```

UJMUVELET OSZTÁLY PÉLDÁNYOSÍTÁSA (MAIN.CPP)

```
/*-----
* UjMuvelet osztály példányosítása
* Demonstrációs célok:
*     -> öröklődés mechanizmusa
*     -> kód újra felhasználás
*     -> őszosztály
*     -> osztályok kiterjesztése
*     -> korai kötés problémája
*     -> inline tagfüggvény
*
* Katona József <katonaj@mail.duf.hu>
*-----*/
#include <iostream> //C++ alapvető adatfolyam I/O rutinok
#include <cstring> //C++ verziója, C "string.h"-nak
#include <conio.h> //DOS konzol I/O rutinok hívása
#include "UjMuvelet.h" //az UjMuvelet osztály deklarációi

using namespace std; //az std névtér behívása

//-----
```

```
int main()
//*****
{
    double a1, b1;
    char m;

    cout << "1. adat: "; cin >> a1;
    cout << "2. adat: "; cin >> b1;

    do
    {
        cout << "muveleti jel (+,-,*,/,^): "; cin >> m;
    }while(!strchr("+-*/*^",m));

    UjMuvelet UjMuvObject(a1,b1,m); // w1 UjMuvelet objektumpéldánya
    UjMuvObject.Vegrehajt();        // aktiválja a Vegrehajt() tagfüggvényt
    UjMuvObject.Kiir();

    _getch();
    return 0;
}
```

ADATOK OSZTÁLY DEKLARÁCIÓJA (ADATOK.H)

```
//*****
* Adatok osztály deklarációja *
* Demonstrációs célok: *
* -> többszörös öröklődés mechanizmusa *
* -> virtualitás *
* *
* Katona József <katonaj@mail.duf.hu> *
*****/
#ifndef ADATOK_H //if not define egy szimbólum nemlétét ellenőrzi
#define ADATOK_H //egy szimbólum definiálása és értékének megadása
//=====
class Adatok {
//=====
protected:
    double x, y;
public:
    Adatok(double x1, double y1);
    ~Adatok(void);
};

#endif //feltétel lezárása
```

EREDMENY OSZTÁLY DEKLARÁCIÓJA (EREDMENY.H)

```

/*****
* Eredmeny osztály deklarációja
* Demonstrációs célok:
*     -> többszörös öröklődés mechanizmusa
*     -> virtualitás
*
* Katona József <katonaj@mail.duf.hu>
*****/
#ifndef EREDMENY_H //if not define egy szimbólum nemlétét ellenőrzi
#define EREDMENY_H //egy szimbólum definiálása és értékének megadása
//=====
class Eredmeny {
//=====
protected:
    double eredmeny;

public:
    Eredmeny(void);
    ~Eredmeny(void);
};
#endif //feltétel lezárása

```

MUVELET OSZTÁLY DEKLARÁCIÓJA (MUVELET.H)

```

/*****
* Muvelet osztály deklarációja
* Demonstrációs célok:
*     -> többszörös öröklődés mechanizmusa
*     -> virtualitás
*
* Katona József <katonaj@mail.duf.hu>
*****/
#ifndef MUVELET_H //if not define egy szimbólum nemlétét ellenőrzi
#define MUVELET_H //egy szimbólum definiálása és értékének megadása
#include "Adatok.h"
#include "MuveletiJel.h"
#include "Eredmeny.h"

//=====
class Muvelet : public Adatok,
               public MuveletiJel,
               public Eredmeny
//=====
{
    ...
}

```

```

public:
    Muvelet(double x1, double y1, char mov1);
    ~Muvelet(void);
    virtual void Szamol();
    void Vegrehajt() { Szamol(); }
    double Ered() { return eredmény; }
    void Kiir();
};
#endif //feltétel lezárása

```

MUVELETIJEL OSZTÁLY DEKLARÁCIÓJA (MUVELETIJEL.H)

```

/*****
 * MuveletiJel osztály deklarációja
 * Demonstrációs célok:
 *     -> többszörös öröklődés mechanizmusa
 *     -> virtualitás
 *
 * Katona József    <katonaj@mail.duf.hu>
 *****/
#ifndef MUVELETIJEL_H //if not define egy szimbólum nemlétét ellenőrzi
#define MUVELETIJEL_H //egy szimbólum definiálása és értékének megadása
//=====
class MuveletiJel {
//=====
protected:
    char mov;

public:
    MuveletiJel(char mov1);
    ~MuveletiJel(void);
};
#endif //feltétel lezárása

```

UjMUVELET OSZTÁLY DEKLARÁCIÓJA (UjMUVELET.H)

```

/*****
 * UjMuvelet osztály deklarációja
 * Demonstrációs célok:
 *     -> többszörös öröklődés mechanizmusa
 *     -> virtualitás
 *
 * Katona József    <katonaj@mail.duf.hu>
 *****/
#ifndef UjMUVELET_H //if not define egy szimbólum nemlétét ellenőrzi
#define UjMUVELET_H //egy szimbólum definiálása és értékének megadása
#include "Muvelet.h"

```

```
//=====
class UjMuvelet : public Muvelet {
//=====
    public:
        UjMuvelet(double x1, double y1, char muv1);
        ~UjMuvelet(void);
        void Szamol();
};
#endif //feltétel lezárása
```

ADATOK OSZTÁLY IMPLEMENTÁLÁSA (ADATOK.CPP)

```
/******
 * Adatok osztály implementálása *
 * Demonstrációs célok: *
 * -> többszörös öröklődés mechanizmusa *
 * -> virtualitás *
 * *
 * Katona József <katonaj@mail.duf.hu> *
 *****/
#include "Adatok.h"

//-----
Adatok::Adatok(double x1, double y1)
//-----
{
    x = x1, y = y1;
}

//-----
Adatok::~Adatok(void){}
//-----
```

EREDMENY OSZTÁLY IMPLEMENTÁLÁSA (EREDMENY.CPP)

```
/******
 * Eredmeny osztály implementálása *
 * Demonstrációs célok: *
 * -> többszörös öröklődés mechanizmusa *
 * -> virtualitás *
 * *
 * Katona József <katonaj@mail.duf.hu> *
 *****/
#include "Eredmeny.h"
```

```
//-----  
Eredmeny::Eredmeny(void)  
//-----  
{  
    eredmeny = 0;  
}  
  
//-----  
Eredmeny::~~Eredmeny(void) {}  
//-----
```

MUVELET OSZTÁLY IMPLEMENTÁLÁSA (MUVELET.CPP)

```
/*-----  
* Muvelet osztály implementálása  
* Demonstrációs célok:  
* -> többszörös öröklődés mechanizmusa  
* -> virtualitás  
*  
* Katona József <katona.jozsef.duf@freemail.hu>  
*-----*/  
#include <iostream>  
#include "Muvelet.h"  
  
using namespace std;  
  
//-----  
Muvelet::Muvelet(double x1, double y1, char mov1):  
    Adatok(x1, y1),  
    MuveletiJel(mov1),  
    Eredmeny()  
//-----  
{  
  
//-----  
Muvelet::~~Muvelet(void) {}  
//-----  
  
//-----  
void Muvelet::Szamol()  
//-----  
{  
    . . . . .  
}
```

```

        switch(muv)
        {
            case '+': eredmény = x + y; break;
            case '-': eredmény = x - y; break;
            case '*': eredmény = x * y; break;
            case '/': eredmény = x / y; break;
            default:
                eredmény = 0;
                cerr << "Ervenytelen muveleti jel: " << muv << endl;
        }
    }

    //-----
    void Muvelet::Kiir()
    //-----
    {
        cout << x << " " << muv << " " << y << " = " << eredmény << endl;
    }

```

MUVELETIJEL OSZTÁLY IMPLEMENTÁLÁSA (MUVELETIJEL.CPP)

```

/*****
* MuveletJel osztály implementálása
* Demonstrációs célok:
*     -> többszörös öröklődés mechanizmusa
*     -> virtualitás
*
* Katona József      <katonaj@mail.duf.hu>
*****/
#include "MuveletiJel.h"

//-----
MuveletiJel::MuveletiJel(char muv1)
//-----
{
    muv = muv1;
}

//-----
MuveletiJel::~MuveletiJel(void) {}
//-----

```

UjMUVELET OSZTÁLY IMPLEMENTÁLÁSA (UjMUVELET.CPP)

```

/*****
 * UjMuvelet osztály implementálása
 * Demonstrációs célok:
 *      -> többszörös öröklődés mechanizmusa
 *      -> virtualitás
 *
 * Katona József      <katona.jozsef.duf@freemail.hu>
 *****/
#include "UjMuvelet.h"
#include <cmath>

//-----
UjMuvelet::UjMuvelet(double x1, double y1, char muv1) : Muvelet(x1, y1, muv1) {}
//-----

//-----
UjMuvelet::~UjMuvelet(void) {}
//-----

//-----
void UjMuvelet::Szamol()
//-----
{
    if (muv == '^')
        eredmeny = pow(x,y);
    else
        Muvelet::Szamol();
}

```

UjMUVELET OSZTÁLY PÉLDÁNYOSÍTÁSA (MAIN.CPP)

```

/*****
 * UjMuvelet osztály példányosítása
 * Demonstrációs célok:
 *      -> többszörös öröklődés mechanizmusa
 *      -> virtualitás
 *
 * Katona József      <katonaj@mail.duf.hu>
 *****/
#include <iostream>
#include <cstring>
#include <conio.h>
#include "UjMuvelet.h"

```

```
using namespace std;

//*****
int main()
//*****
{
    double a1, b1;
    char m;

    cout << "1. adat: "; cin >> a1;
    cout << "2. adat: "; cin >> b1;

    do
    {
        cout << "muveleti jel (+,-,*,/,^): "; cin >> m;
    }while(!strchr("+-*/^",m));

    UjMuvelet UjMuvObject(a1,b1,m);
    UjMuvObject.Vegrehajt();
    UjMuvObject.Kiir();

    _getch();
    return 0;
}
```

4. Felhasznált irodalom

- Andrei Alexandrescu–Herb Sutter(2005): *C++ kódolási szabályok*. Kiskapu kiadó.
- Robert A. Maksimchuck–Eric J. Naiburg (2006): *UML földi halandóknak*. Kiskapu kiadó.
- Alex Allain (2013): *Jumping Into C++*. Cprogramming.com kiadó.
- Mike McGrath(2011): *C++ Programming In Easy Steps 4th Edition*. In Easy Steps Limited kiadó.
- Bjarne Stroustrup (2013): *The C++ Programming Language*. Addison Wesley kiadó
- Nicolai M. Josuttis (2012): *The C++ Standard Library: A Tutorial and Reference*. Addison Wesley kiadó.
- Ivor Horton (2014): *Ivor Horton's Beginning Visual C++ 2013* (Wrox Beginning Guides). John Wiley & Sons kiadó.
- David Vandevor (2014): *C++ Templates: The Complete Guide*. Addison Wesley kiadó.
- Marc Gregoire (2014): *Professional C++*. John Wiley & Sons kiadó.
- Martin Fowler(2003): *UML Distilled: A Brief Guide to the Standard Object Modeling Language* (Object Technology Series). Addison Wesley kiadó.
- Craig Larman(2004): *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Prentice Hall kiadó.
- Simon Bennet–Steve Mcrobb–Ray Farmer (2006): *Object-Oriented Systems Analysis and Design Using UML*. McGraw-Hill Higher Education kiadó.
- David P. Tegarden–Alan Dennis–Barbara Haley Wixom (2012): *Systems Analysis and Design with UML*. John Wiley & Sons kiadó.

Segédanyagok

Továbbá számos olyan nagyobb magyar és külföldi egyetemektől származó publikáció, könyv és segédlet került felhasználásra, amelyek külön nem kaptak helyet a felsorolásban.

