Objektumorientált programozás

Objektumalapú programozás a C++ programozási nyelvben

Az objektumorientált programozási módszer

Darvay Zsolt

Túlterhelés, konverzió és alosztályok

14. Operátorok túlterhelése
15. A programozó által definiált típuskonverzió
16. Az objektumorientált programozási módszer

16. Az objektumorientált programozási módszer

Származtatott osztályok

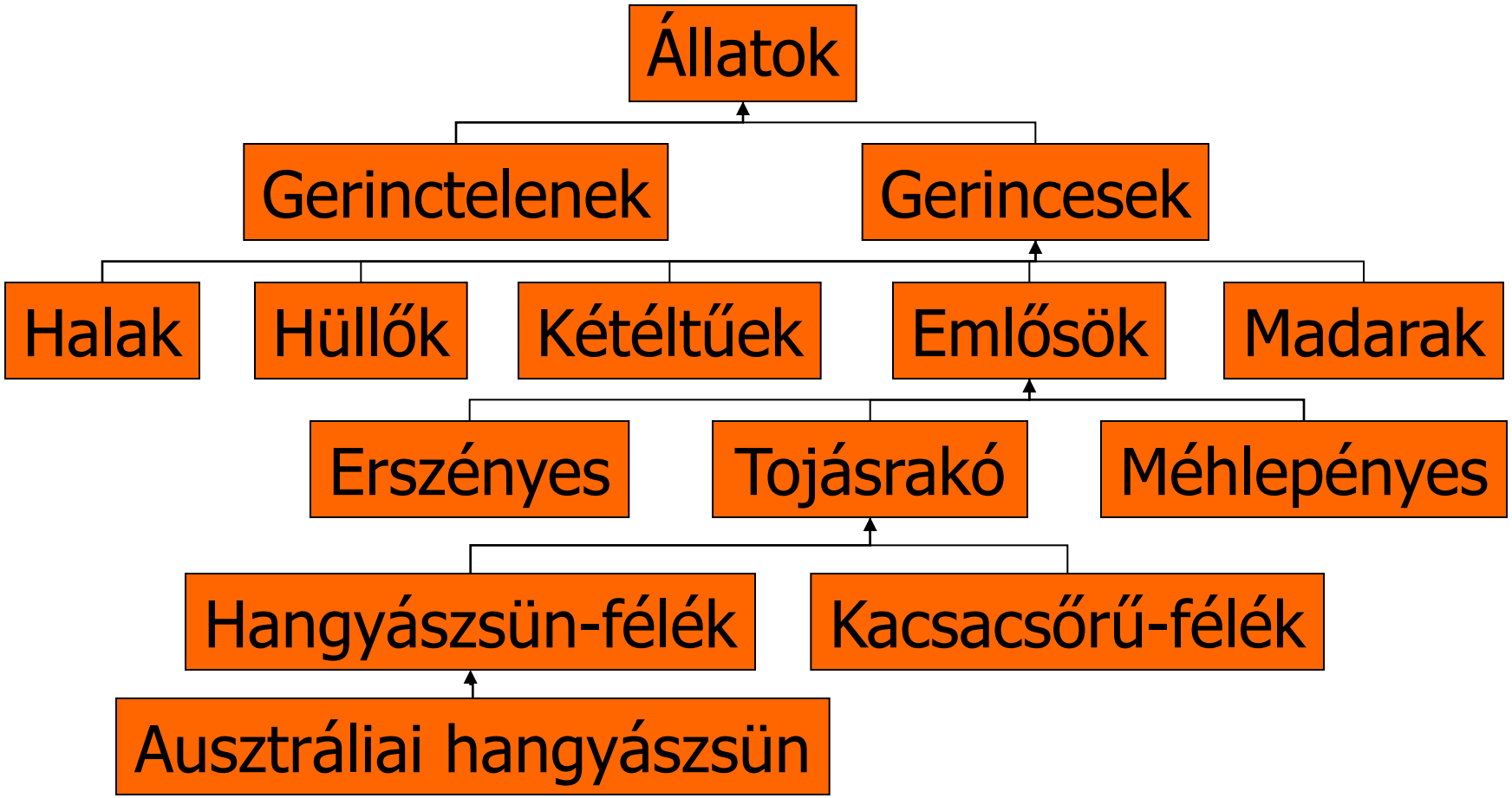
Áttekintés

- 16.1. Elméleti alapok
- 16.2. Származtatott osztályok deklarációja
- 16.3. Virtuális tagfüggvények
- 16.4. Virtuális osztályok
- 16.5. Absztrakt osztályok. A tiszta virtuális tagfüggvény

16.1. Elméleti alapok

- Az objektum adattagokat és tagfüggvényeket tartalmaz.
- A védett tagok csak a tagfüggvényekben érhetők el (nem használunk barát függvényeket).
- Ezt a tulajdonságot egybezártságnak (zártságnak) nevezzük.
- A gyakorlatban nem csak különálló objektumokkal találkozunk. A különböző objektumok közti kapcsolatok is fontosak.

Állatok


Állatok
Gerinctelenek
Gerincesek
Halak
Hüllők
Kétéltűek
Emlősök
Madarak
Erszényes
Tojásrakó
Méhlepényes
Hangyászsün-félék
Kacsacsőrű-félék
Ausztráliai hangyászsün

Az öröklés

- Egy osztály örökölheti egy másik osztály tagjait.
- Az eredeti osztály neve alaposztály (vagy bázisosztály). Az örökléssel létrehozott osztályt származtatott osztálynak nevezzük.
- Az adattagok, és a tagfüggvények is öröklődnek.
- Ha egy oszály több alaposztállyal rendelkezik, akkor többszörös öröklésről beszélünk.
- Az öröklés egy másik fontos tulajdonsága az objektumoknak. Az objektumok egy hierarchiát alkothatnak.

A polimorfizmus

- Az öröklött tagfüggvények túlterhelhetőek.
- Nem csak a függvény neve, hanem a paraméterlistája is ugyanaz lehet.
- Az objektumhierarchia különböző szintjein ugyanannak a műveletnek más és más értelme lehet.
- Ezt a tulajdonságot polimorfizmusnak (többalakúságnak) nevezzük.

16.2. Származtatott osztályok deklarációja

```
class oszt : alaposztálylista {
        // új adattagok és tagfüggvények
};
```

- Az alaposztálylista vesszővel elválasztott elemekből áll, amelyek a következők lehetnek:
  - public alaposztály
  - protected alaposztály
  - private alaposztály

Példa (az oszt_1 osztály)

```
#include <iostream>
using namespace std;
class oszt_1 {
private:
   int x;
public:
   oszt_1(/*paraméterlista_1*/);
};
```

Az oszt_n osztály

```
class oszt_n {
protected:
   int y;
public:
   oszt_n(/*paraméterlista_n*/);
};
```

Konstruktorok

```
oszt_1::oszt_1(/*paraméterlista_n*/)
{
   x = 10;
}
oszt_n::oszt_n(/*paraméterlista_n*/)
{
   y = 100;
}
```

Példa alaposztálylistára

```
class oszt : public oszt_1, /* ..., */ public oszt_n {
public:
   oszt(/*paraméterlista*/);
   void kiir();
};
```

- Az oszt osztály az oszt_1, ..., oszt_n osztályok származtatott osztálya.

A tagok elérése a származtatott osztályban

Az alaposztályban	Az alaposztálylistabeli hozzáférésmódosító	A származtatott osztályban
public	public	public
protected	public	protected
private	public	nem elérhető
public	protected	protected
protected	protected	protected
private	protected	nem elérhető
public	private	private
protected	private	private
private	private	nem elérhető

A származtatott osztály konstruktora

- A konstruktorok és destruktorok nem öröklődnek.
- A származtatott osztály konstruktorának definiálása:

```
oszt::oszt(/*paraméterlista*/) :
   oszt_1(/*lista1*/), /*...,*/ oszt_n(/*lista_n*/)
{
        // ...
}
```

Elérhetőség (a kiir metódus)

```
void oszt::kiir()
{
   // cout << x << endl;  // nem elérhető (private)
   cout << y << endl;
}
```

Elérhetőség (main)

```
int main()
{
   oszt ob;
   ob.kiir();// kiírja a 100-at
   //ob.y = 1000;     // nem elérhető (protected)
}
```

2. Példa származtatott osztályra

```
#include <iostream>
using namespace std;
class alap {                    // az alaposztály
public:
   void f1();
   void f2();
};
```

A származtatott osztály

```
class szarm : public alap {
public:
   void f1();
};
```

- Csak az f1 tagfüggvényt terheljük túl.
- Az f2 öröklődik az alaposztályból.

Az alaposztály tagfüggvényei

```
void alap::f1()
{
   cout << "alap: f1\n";
}
void alap::f2()
{
   cout << "alap: f2\n";
   f1();                  // az f2 meghívja az f1-et.
}
```

A származtatott osztály tagfüggvénye

```
void szarm::f1()
{
   cout << "szarmaztatott: f1\n";
}
```

Statikus kötés

```
int main() {
   szarm s;
   s.f2();
}
```

Kimenet:

```
alap: f2
alap: f1
```

- Az f1 függvény kiválasztása fordítási időben történt, ezért az alaposztály f1 tagfüggvénye lesz végrehajtva. Ezt a tulajdonságot statikus kötésnek nevezzük.

Dinamikus kötés

- Ha a végrehajtandó függvény kiválasztása futási időben történik, akkor dinamikus kötésről beszélünk.
- A dinamikus kötést virtuális tagfüggvények segítségével valósíthatjuk meg.
- Az f1 tagfüggvényt kell virtuálisnak deklarálni. Ezt úgy tehetjük meg, hogy a virtual minősítőt használjuk a függvény alaposzálybeli deklarációjában.

16.3. Virtuális tagfüggvények

```
class alap {          // az alaposztály
public:
   virtual void f1();
   void f2();
};
```

- A virtual kulcsszót elég egyszer megadni, az alaposztálybeli deklarációban.
- Ebben az esetben a származtatott osztályban deklarált túlterhelt tagfüggvény is virtuális lesz.

A fő függvény

```
int main() {
   szarm s;
   s.f2();
}
```

Kimenet:

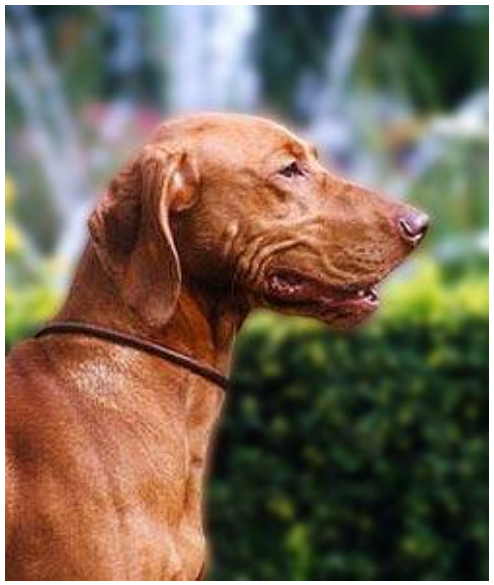
alap: f2

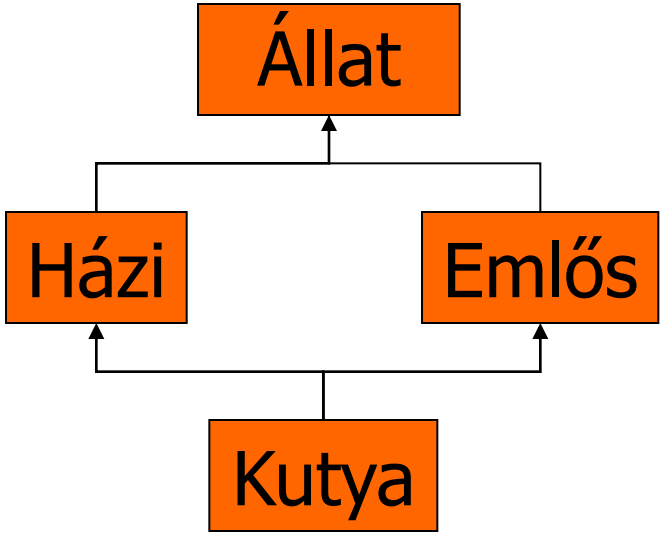
szarmaztatott: f1

- Ha egy függvényt virtuálisnak deklaráltunk az alaposztályban, akkor az osztályhierarchia tetszőleges származtatott osztályában virtuális lesz.

16.4. Virtuális osztályok

- Többszörös öröklés esetén egy származtatott osztály több példányban is örökölhet egy adattagot.





Az állat osztály

```
#include <iostream>
#include <cstring>
using namespace std;
class allat {
protected:
   char nev[20];
public:
   allat(const char* n);
};
```

Az „emlos” osztály

```
class emlos : public allat {
protected:
   int suly;
public:
   emlos(const char* n, int s);
};
```

A „hazi” osztály

```
class hazi : public allat {
protected:
   int viselkedes;
public:
   hazi(const char* n, int v);
};
```

A kutya osztály

```
class kutya : public emlos, public hazi {
protected:
   bool ugat;
public:
   kutya(const char* n, int s, int v, bool u);
   void kiir();
};
```

Az „allat” osztály konstruktora

```
allat::allat(const char* n)
{
   strcpy(nev, n);
}
```

Az „emlos” és „hazi” osztályok konstruktorai

```
emlos::emlos(const char* n, int s) : allat(n)
{
   suly = s;
}
hazi::hazi(const char* n, int v) : allat(n)
{
   viselkedes = v;
}
```

A kutya osztály konstruktora

```
kutya::kutya(const char* n, int s, int v, bool u) :
   emlos(n, s), hazi(n, v)
{
   ugat = u;
}
```

A kiir tagfüggvény

```
void kutya::kiir()
{
   cout << "nev (emlos): " << emlos::nev << endl;
   cout << "nev (hazi): " << hazi::nev << endl;
   cout << "suly: " << suly << endl;
   cout << "viselkedes: " << viselkedes << endl;
   if (ugat) cout << "ugat\n";
   else cout << "nem ugat\n";
}
```

A fő függvény

```
int main() {
   kutya v("magyar vizsla", 12, 9, true);
   v.kiir();
}
```

- A kiir tagfüggvényben nem lehet egyszerűen a nev segítségével hivatkozni a kutya nevére, mivel ez az adattag két különböző ágon keresztül öröklődik.

A kimenet

nev (emlos): magyar vizsla
nev (hazi): magyar vizsla
suly: 12
viselkedes: 9
ugat

- A nev adattag két példányban lesz jelen a kutya osztályban, ezért az alaposztály nevével és a hatókör operátorral hivatkozunk rá. Ellenkező esetben a fordító hibát jelezne.

Virtuális bázisosztály

- Ha azt szeretnénk, hogy a nev adattag csak egy példányban legyen jelen a kutya osztályban, akkor virtuális osztályokat kell használni.
- Egy osztály az öröklésкor válhat virtuálissá.
- Ennek érdekében a virtual kulcsszót kell elhelyezni az alaposztálylistában, az osztály neve elé.
- Ekkor az alaposztály virtuális lesz az illető származtatott osztályra nézve.

Az „emlos” osztály

```
class emlos : public virtual allat {
protected:
   int suly;
public:
   emlos(const char* n, int s);
};
```

A „hazi” osztály

```
class hazi : public virtual allat {
protected:
   int viselkedes;
public:
   hazi(const char* n, int v);
};
```

A kutya osztály konstruktora

```
kutya::kutya(const char* n, int s, int v, bool u) :
   emlos::allat(n), emlos(n, s), hazi(n, v)
{
   ugat = u;
}
```

- A virtualitás miatt az „emlos” és a „hazi” osztályok konstruktorai nem hívják meg az „allat” osztály konstruktorát, ezért ezt nekünk kell megtenni.

A kiir tagfüggvény

```
void kutya::kiir()
{
   cout << "nev: " << nev << endl;
   cout << "suly: " << suly << endl;
   cout << "viselkedes: " << viselkedes << endl;
   if (ugat) cout << "ugat\n";
   else cout << "nem ugat\n";
}
```

A fő függvény

```
int main() {
   kutya v("magyar vizsla", 12, 9, true);
   v.kiir();
}
```

- A kiir tagfüggvényben egyszerűen a nev segítségével hivatkozhatunk a kutya nevére.

A kimenet

nev: magyar vizsla

suly: 12

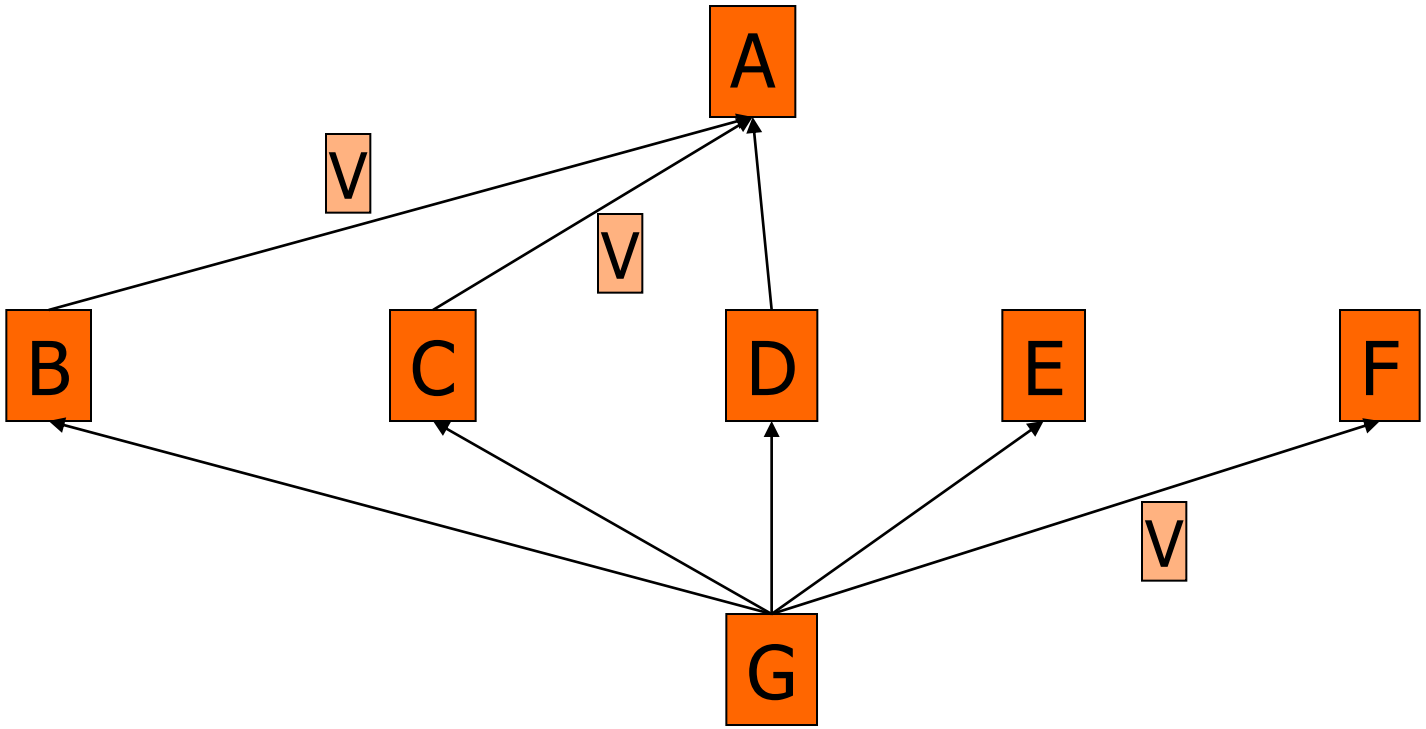
viselkedes: 9

ugat

Öröklődés, virtuális és nem virtuális módon is

- Egy bonyolult osztályhierarchia esetén egy alaposztályt egyidejűleg virtuális és nem virtuális módon is örökölhet egy származtatott osztály.

Példa



- A V jelöli a virtuális öröklődést.

A konstruktorok meghívásának sorrendje

- A származtatott osztály objektumának létrehozásakor előbb a virtuális alaposztályok konstruktorait hívja meg a rendszer az alaposztálylistának megfelelő sorrendben.
- Ezt követően a nem virtuális alaposztályok konstruktorai lesznek végrehajtva, ugyancsak az alaposztálylistának megfelelő sorrendben.

A konstruktormeghívások száma

- Ha egy alaposztályt virtuális és nem virtuális módon is örököl egy származtatott osztály, akkor az összes virtuális példányra egyszer lesz végrehajtva a konstruktora, és még annyiszor, ahány nem virtuális példány van.

Az „A” osztály

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

class A {
public:
   A(char* s) { strcat(s, "A"); }
};
```

A „B” és „C” osztályok

```
class B : virtual public A {
public:
   B(char* s) : A(s) { strcat(s, "B"); }
};
class C : virtual public A {
public:
   C(char* s) : A(s) { strcat(s, "C"); }
};
```

A „D” osztály

```
class D : public A {
public:
   D(char* s) : A(s) { strcat(s, "D"); }
};
```

Az „E” és „F” osztályok

```
class E {
public:
   E(char* s) { strcat(s, "E"); }
};
class F {
public:
   F(char* s) { strcat(s, "F"); }
};
```

A „G” osztály

```
class G : public B, public C, public D,
          public E, virtual public F {
public:
   G(char* s) : B::A(s), B(s), C(s), D(s), E(s), F(s) {
        strcat(s, "G");
   }
};
```

- Ha a B:: hiányozna, akkor ambiguous access of 'A' hibaüzenet jelenne meg.

A fő függvény

```
int main() {
   char s[20];
   strcpy(s, "");
   G g(s);
   cout << s << endl;
}
```

Kimenet:

AFBCADEG

16.5. Absztrakt osztályok. A tiszta virtuális tagfüggvény

- Egy alaposztálynak lehetnek olyan általános tulajdonságai, amelyekről tudunk, de nem tudjuk őket definiálni csak egy származtatott osztályban.
- Ebben az esetben egy olyan virtuális tagfüggvényt deklarálhatunk, amely nem lesz definiálva az alaposztályban. Azokat a tagfüggvényeket, amelyek deklarálva vannak, de nincsenek definiálva egy adott osztályban, tiszta virtuális tagfüggvényeknek nevezzük.

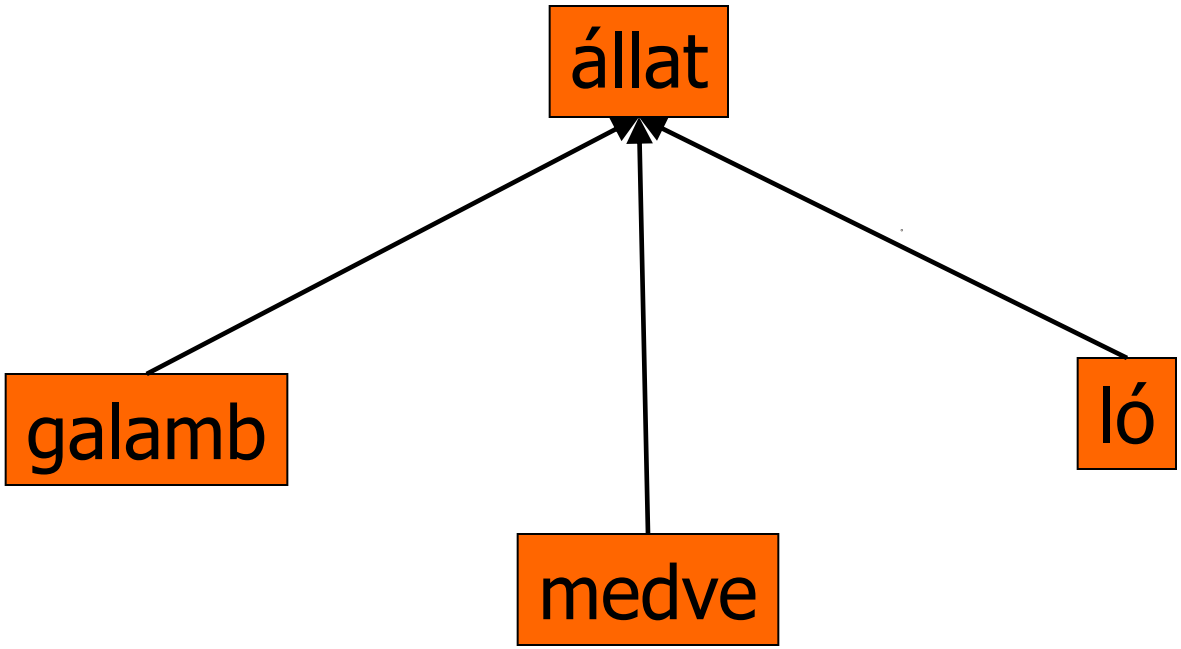
A tiszta virtuális tagfüggvény deklarálása

- A tiszta virtuális tagfüggvényt a szokásos módon deklaráljuk, de a fejléc után az =0 karaktereket írjuk. Ez jelzi, hogy a tagfüggvényt nem fogjuk definiálni.
- Azokat az osztályokat, amelyek tartalmaznak legalább egy tiszta virtuális tagfüggvényt, absztrakt osztályoknak nevezzük.
- Az absztrakt osztályoknak nem hozhatjuk létre objektumát.

A tiszta virtuális tagfüggvények túlterhelése

- A tiszta virtuális tagfügvényeket túl kell terhelni a származtatott osztályban, ellenkező esetben az illető osztály is absztrakt lesz.

Példa



Az „allat” osztály adattagjai

```
#include <iostream>
using namespace std;
class allat {
protected:
   double suly;              // kg
   double eletkor;           // ev
   double sebesseg;          // km / h
public:
   // ...
```

Az „allat” osztály tagfüggvényei

```
    allat(double su, double k, double se);
    virtual double atlagos_suly() = 0;
    virtual double atlagos_eletkor() = 0;
    virtual double atlagos_sebesseg() = 0;
    int kover() { return suly > atlagos_suly(); }
    int gyors() { return sebesseg > atlagos_sebesseg(); }
    int fiatal() { return 2 * eletkor < atlagos_eletkor(); }
    void kiir();
};
```

Az „allat” osztály konstruktora

```
allat::allat(double su, double k, double se)
{
    suly = su;
    eletkor = k;
    sebesseg = se;
}
```

Kiírás

```
void allat::kiir()
{
    cout << (kover() ? "kover, " : "sovany, ");
    cout << (fiatal() ? "fiatal, " : "oreg, ");
    cout << (gyors() ? "gyors" : "lassu") << endl;
}
```

A „galamb” osztály

```
class galamb : public allat {
public:
    galamb(double su, double k, double se) :
        allat(su, k, se) {}
    double atlagos_suly() { return 0.5; }
    double atlagos_eletkor() { return 6; }
    double atlagos_sebesseg() { return 90; }
};
```

A „medve” osztály

```
class medve : public allat {
public:
    medve(double su, double k, double se) :
        allat(su, k, se) {}
    double atlagos_suly() { return 450; }
    double atlagos_eletkor() { return 43; }
    double atlagos_sebesseg() { return 40; }
};
```

A „lo” osztály

```
class lo : public allat {
public:
    lo(double su, double k, double se) :
        allat(su, k, se) {}
    double atlagos_suly() { return 1000; }
    double atlagos_eletkor() { return 36; }
    double atlagos_sebesseg() { return 60; }
};
```

A fő függvény

```
int main() {
    galamb g(0.6, 1, 80);
    medve m(500, 40, 46);
    lo l(900, 8, 70);
    g.kiir();
    m.kiir();
    l.kiir();
}
```

A kimenet

kover, fiatal, lassu

kover, oreg, gyors

sovany, fiatal, gyors