



**Pázmány Péter Katolikus Egyetem
Információs Technológiai és Bionikai Kar**

Bevezetés a Programozásba II

7. előadás

Öröklődés

© 2014.03.24. Giachetta Roberto

groberto@inf.elte.hu

<http://people.inf.elte.hu/groberto>

Öröklődés

Példa

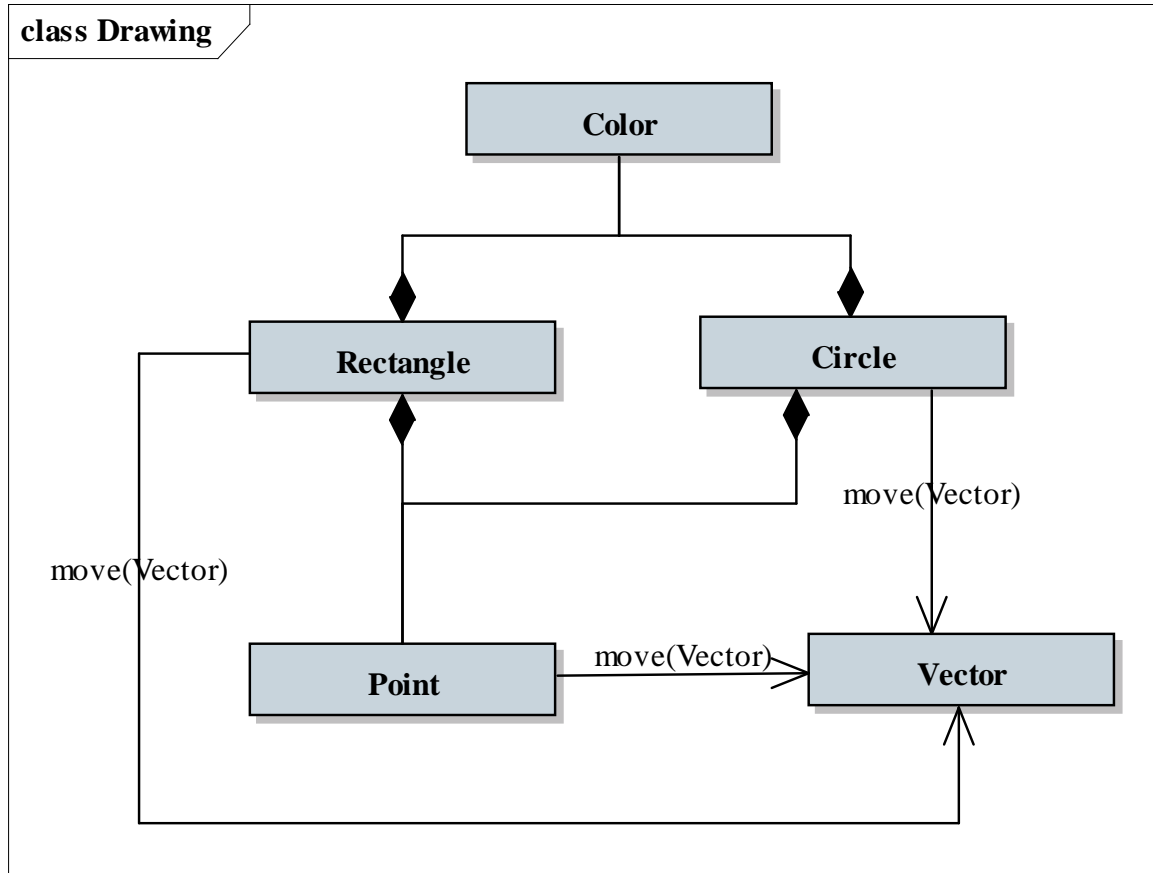
Feladat: Készítsünk egy programot, amelyben téglalapokat és köröket tudunk a képernyőre rajzolni, amelyek három színben (piros, fehér, zöld) váltakoznak. A színeket utólag le tudjuk cserélni (új véletlenszerű színre), és a teljes rajzot el tudjuk tolni egy vektorral valamilyen irányba.

- a korábbi téglalap (**Rectangle**) mellé felvesszünk egy új alakzatot, a kört (**Circle**), szóközzel váltunk közöttük
- a kört középpontjával (**_point**), illetve sugarával (**_radius**) ábrázoljuk, megadjuk rá a megfelelő műveleteket (**draw**, **contains**, **move**)
- a főprogramban felvesszünk egy körökből álló vektort (**circs**), és azon is végrehajtunk mindent

Öröklődés

Példa

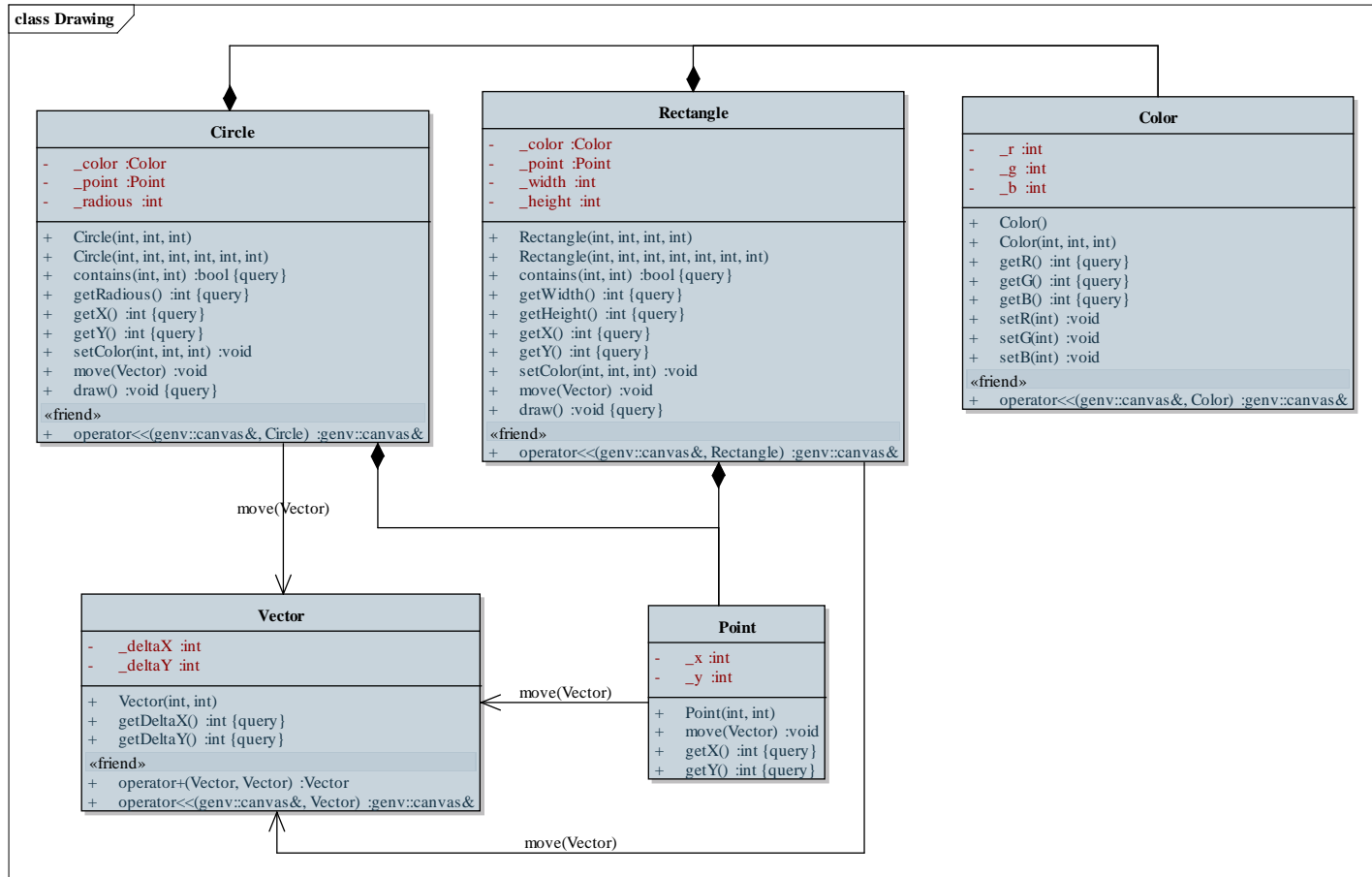
Tervezés:



Öröklődés

Példa

Tervezés:



Öröklődés

Kódismétlődés objektum-orientált szerkezetben

- Az objektum-orientált programokban az osztályok definiálják az objektumok működési sémáját
 - különböző osztályok részei megegyezhetnek (hasonló működés, reprezentáció), ami *kódismétlődés*hez vezethet
 - pl.: az egyetemi oktató és az egyetemi hallgató is tartalmaz nevet, azonosítót, kurzuslistát
- Az osztályok közötti kódismétlődésnek két esete van:
 - több osztály rendelkezik közös részekkel
 - egy osztály rendelkezik egy másik osztály valamennyi tulajdonságával, ekkor az osztály *speciális esete* a másiknak (a másik pedig *általánosítása* az egyiknek)

Öröklődés

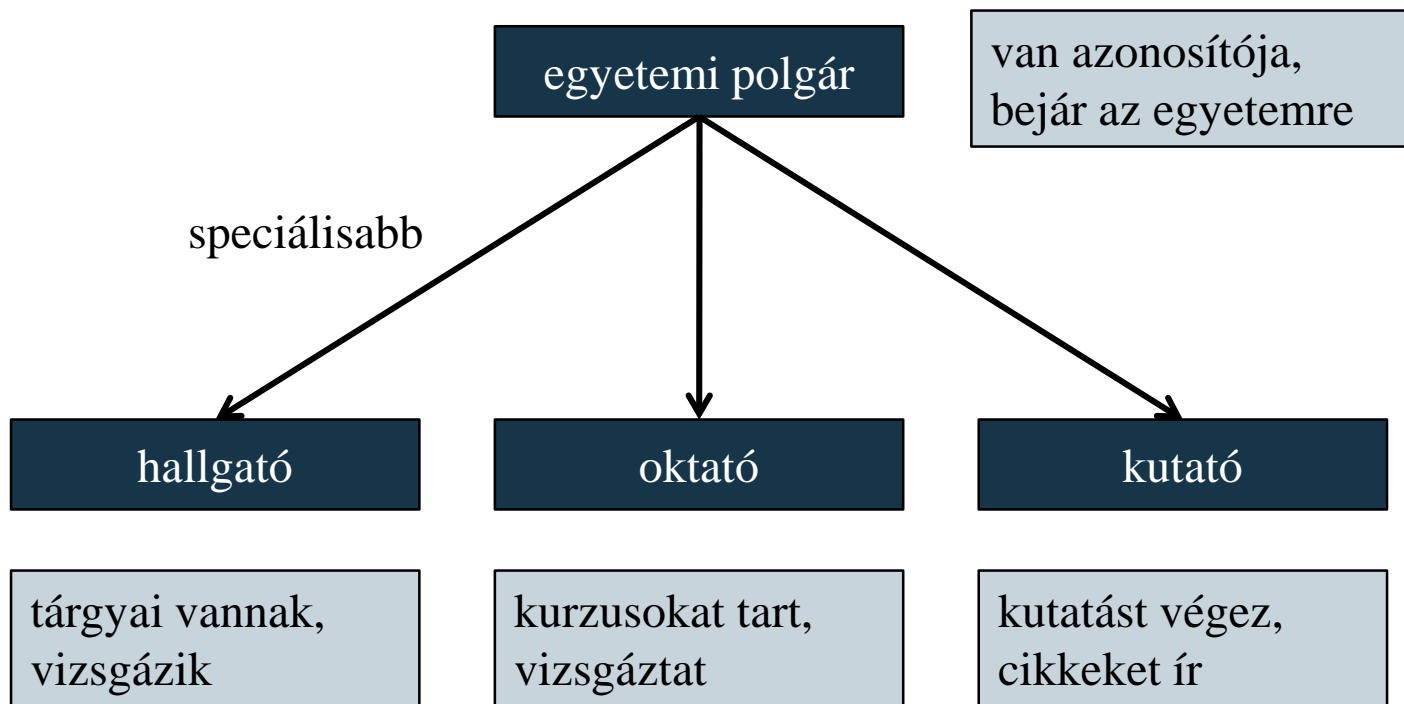
Általánosítás és specializáció

- Az első eset kombinálható a másodikkal, hiszen ha több osztálynak közös tulajdonságai vannak, akkor van olyan általánosabb osztály, amely azokat a tulajdonságokat tartalmazza
 - pl. az egyetemi hallgató és oktató közös tulajdonságait az egyetemi polgár osztály tartalmazza, így azok ennek speciálisabb változatai lesznek
- Egy osztálynak tehát lehet egy, vagy több speciálisabb változata, amely mindent tud, amit az általánosabb, és ezen felül még kiegészítheti azt tetszőleges módon
 - fordítva, egy osztálynak lehet egy (esetekben akár több) általánosabb változata, amelytől átveszi a viselkedést

Öröklődés

Általánosítás és specializáció

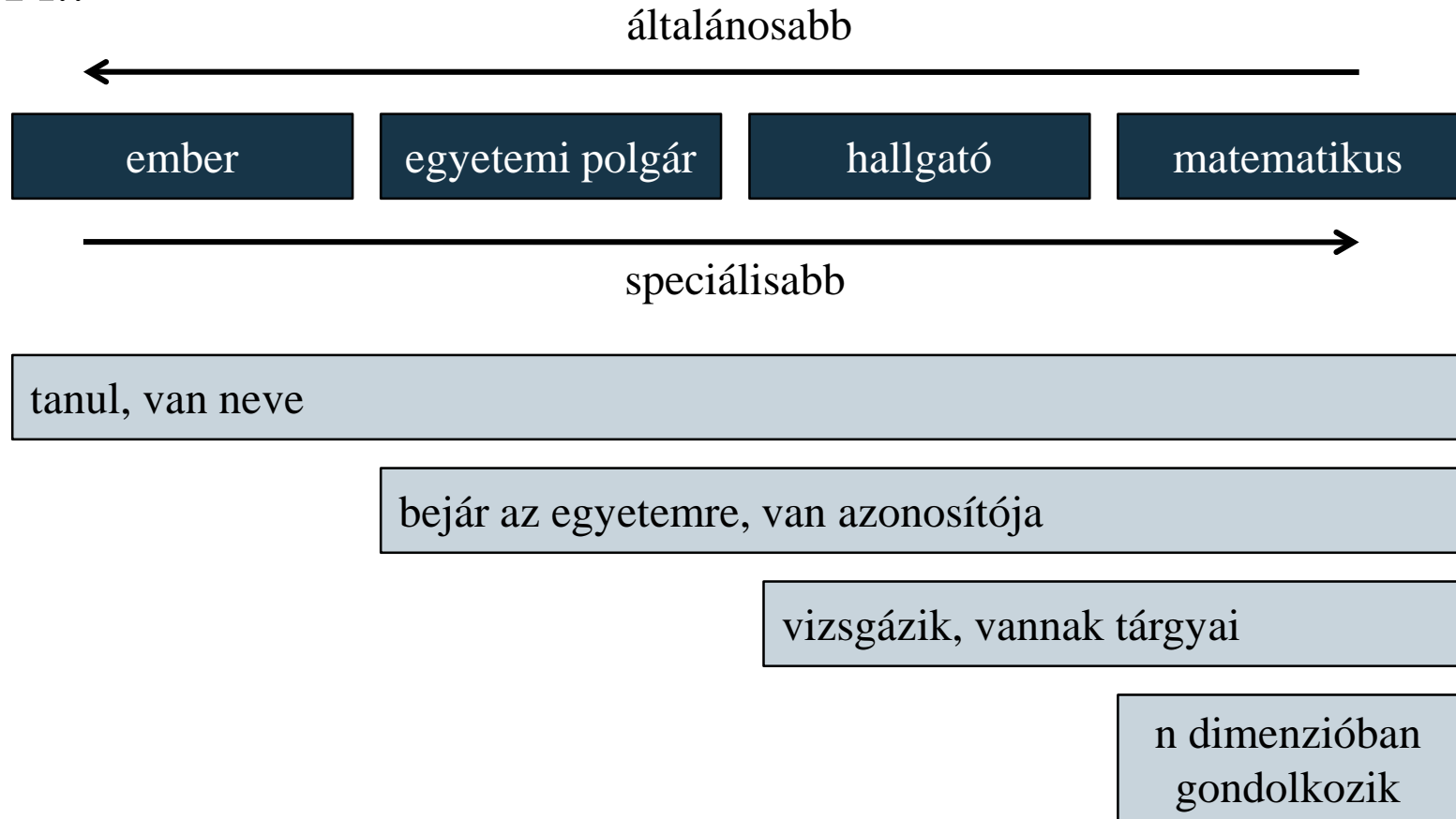
- Pl.:



Öröklődés

Általánosítás és specializáció

- Pl.:



Öröklődés

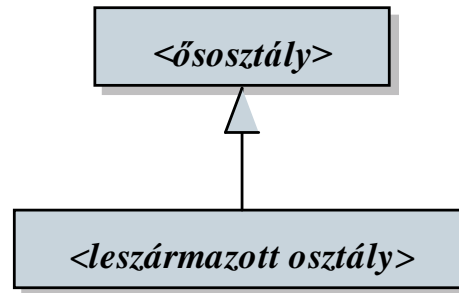
Specializáció és általánosítás

- Azt a folyamatot, amikor a speciálisabb átveszi az általános jellemzőit és működését, *öröklődésnek* (vagy *származtatásnak*) nevezzük
- Az öröklődés két iránya a *specializáció*, és az *általánosítás*
 - az általánosabb objektumot *ősnek* (*base*), a speciálisabbat *leszármazottnak* (*descendant*) nevezzük (ha csak egy szint a különbség, akkor *szülőnek*, illetve *gyereknek*)
 - egy osztálynak több gyereke, leszármazottja, illetve őse lehet
 - ha egy osztálynak több szülője is van, akkor *többszörös öröklődésről* beszélünk (ez ritka, bizonyos esetekben tiltott)

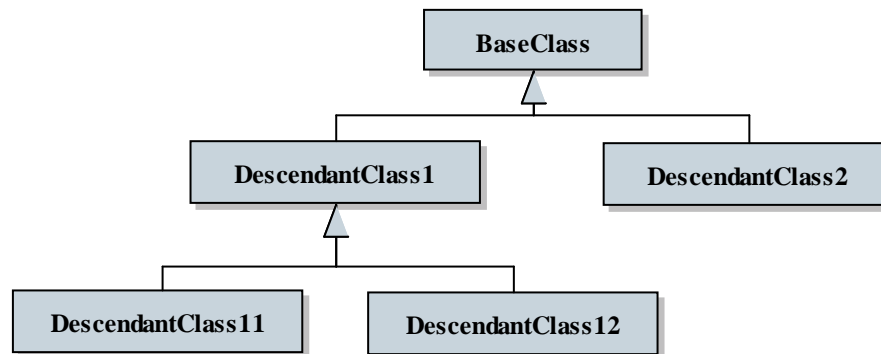
Öröklődés

Jelölése

- Az osztálydiagramban az öröklődést megfelelő relációval jelölhetjük:



- Az öröklődések sorozata előállít egy *öröklődési fát* (többszörös öröklődés esetén *öröklődési gráfot*):



Öröklődés

Megvalósítása

- Az öröklődést az osztályoknál kell jelölünk úgy, hogy megadjuk, mely őosztály(ok)nak örökli a tulajdonságait

- az öröklődés is rendelkezik láthatósággal

- egyszeres öröklődés:

```
class <osztálynév> : <láthatóság> <ős> {  
    <osztályfelület>  
};
```

- többszörös öröklődés:

```
class <osztálynév> : <láthatóság> <ős 1>,  
                  <láthatóság> <ős 2>, ... {  
    <osztályfelület>  
};
```

Öröklődés

Megvalósítása

- Pl.:

```
class MyBaseClass {  
    // általános osztály  
public:  
    int _base; // mező  
    MyBaseClass() { _base = 1; } // konstruktor  
    void setBaseValue(int v) { _base = v; }  
    // metódus  
};
```

```
MyBaseClass mbc;  
cout << mbc._base; // kiírja: 1  
mbc.setBaseValue(5);  
cout << mbc._base; // kiírja: 5
```

Öröklődés

Megvalósítása

- Pl.:

```
class MyChildClass : public MyBaseClass {
    // speciális osztály, a korábbi leszármazottja,
    // automatikusan megkapja a _base,
    // setBaseValue tagokat
public:
    int _child; // további mező
    MyChildClass() { _child = 2; }
    // konstruktor
    void childMethod() { // további metódus
        _base = _child;
        // az örökölt tagok felhasználhatóak
    }
};
```

Öröklődés

Megvalósítása

- Pl.:

```
MyChildClass mcc; // leszármazott példányosítása
```

```
// elérhetjük az örökölt tagokat:
```

```
cout << mcc._base; // kiírja: 1
```

```
mcc.setBaseValue(5);
```

```
cout << mcc._base; // kiírja: 5
```

```
// elérhetjük az új tagokat:
```

```
cout << mcc._child; // kiírja: 2
```

```
mcc.childMethod();
```

```
cout << mcc._base; // kiírja: 2
```

Öröklődés

Láthatóság használata

- Az osztálytulajdonságokra 3 láthatóságot alkalmazhatunk:
 - **public (+)**: látható, öröklődik
 - **private (-)**: rejtett, öröklődik, azonban a leszármazott osztályokban nem lesz látható
 - **protected (#)**: védett, öröklődik, az összes leszármazott osztályban látható lesz
- Magára az öröklődésre is három láthatóságot alkalmazhatunk:
 - **public**: minden a megadott láthatósággal öröklődik tovább
 - **protected**: a látható tagok védetté válnak
 - **private**: a látható és védett tagok rejtetté válnak

Öröklődés

Láthatóság használata

- Pl.:

```
class MyBaseClass {  
private: // most már rejtett lesz a mező  
    int _base;  
public:  
    MyBaseClass() { _base = 1; }  
    void setBaseValue(int v) { _base = v; }  
    int getBaseValue() { return _base; }  
};
```

```
MyBaseClass mbc;  
cout << mbc.getBaseValue(); // kiírja: 1  
mbc.setBaseValue(5);  
cout << mbc.getBaseValue(); // kiírja: 5
```


Öröklődés

Láthatóság használata

- Pl.:

```
class MyChildClass : public MyBaseClass {
    // a _base már nem látszódik, de öröklődik
private:
    int _child;
public:
    MyChildClass() { _child = 2; }
    void childMethod() {
        setBaseValue(_child);
        // nem látja a _base mezőt, de közvetetten
        // használhatja
    }
    int getChildValue() { return _child; }
};
```

Öröklődés

Láthatóság használata

- Pl.:

```
MyChildClass mcc; // leszármazott példányosítása
```

```
// elérhetjük az örökölt tagokat:
```

```
cout << mcc.getBaseValue(); // kiírja: 1
```

```
mcc.setBaseValue(5);
```

```
cout << mcc.getBaseValue(); // kiírja: 5
```

```
// elérhetjük az új tagokat:
```

```
cout << mcc.getChildValue(); // kiírja: 2
```

```
mcc.childMethod();
```

```
cout << mcc.getBaseValue(); // kiírja: 2
```

Öröklődés

Láthatóság használata

- Pl.:

```
class MyBaseClass {
protected: // most védett lesz a mező
    int _base;
public:
    MyBaseClass() { _base = 1; }
    void setBaseValue(int v) { _base = v; }
    int getBaseValue() { return _base; }
};
```

```
MyBaseClass mbc;
cout << mbc.getBaseValue(); // kiírja: 1
mbc.setBaseValue(5);
cout << mbc.getBaseValue(); // kiírja: 5
```

Öröklődés

Láthatóság használata

- Pl.:

```
class MyChildClass : public MyBaseClass {  
    // minden elérhető lesz az ősből  
private:  
    int _child;  
public:  
    MyChildClass() { _child = 2; }  
    void childMethod() {  
        _base = _child;  
        // látható lesz a leszármazottban, de kívül  
        // nem  
    }  
    int getChildValue() { return _child; }  
};
```

Öröklődés

Példa

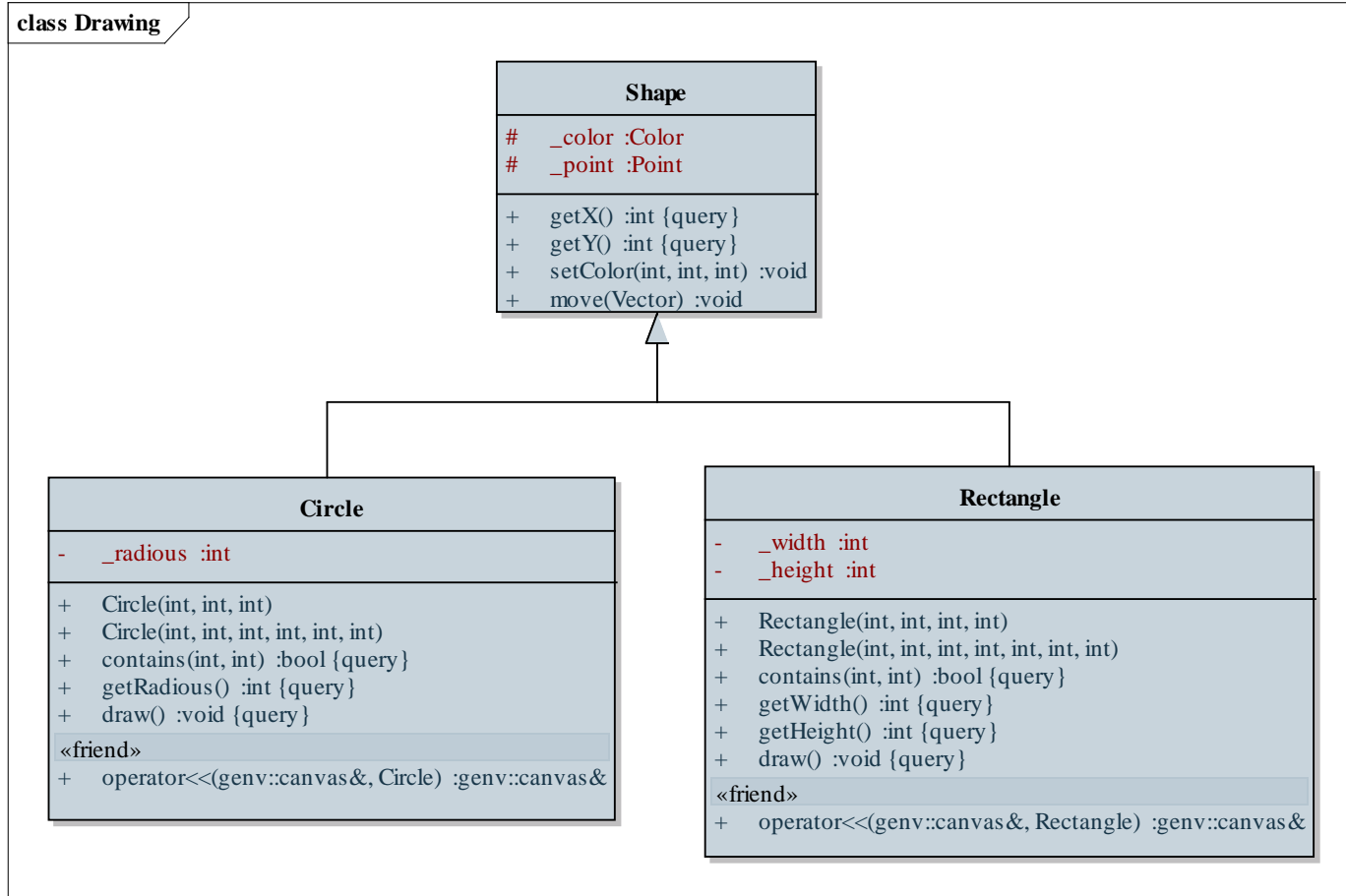
Feladat: Készítsünk egy programot, amelyben téglalapokat és köröket tudunk a képernyőre rajzolni, amelyek három színben (piros, fehér, zöld) váltakoznak. A színeket utólag le tudjuk cserélni (új véletlenszerű színre), és a teljes rajzot el tudjuk tolni egy vektorral valamilyen irányba.

- javítsunk a korábbi megoldáson öröklődés segítségével, egy közös osztályt (**shape**) hozunk létre, amelybe helyezzük a közös adatokat (**_point**, **_color**), ezeknek védett (**protected**) láthatóságot adunk
- az ősből elvégezhető műveleteket áthelyezzük (**move**, **getX**, **getY**), így megszűnik az osztályokban a kódismétlődés

Öröklődés

Példa

Tervezés:



Öröklődés

Tagok elrejtése és elérése

- Az őssel megegyező szintaktikájú metódusok *elrejtik* az örökölt tulajdonságokat, azaz híváskor a leszármazott osztály megfelelő tulajdonságát érjük el
 - lehetőségünk van explicit hivatkozni az ős bármely látható tulajdonságára az **<ős osztály>::** előtaggal (csak az azonos nevű mezők, illetve azonos szintaktikájú metódusoknál szükséges)

- Pl.:

```
class MyBaseClass {  
public:  
    void PrintValue() { cout << 1 << endl; }  
};
```

Öröklődés

Tulajdonságok elrejtése és elérése

```
class MyChildClass : public MyBaseClass {
public:
    void PrintValue() { cout << 2 << endl; }
    // elrejtő metódus
    void PrintBaseValue() {
        MyBaseClass::PrintValue();
    } // ős metódusának meghívása
};
```

...

```
MyBaseClass mbc; mbc.PrintValue(); // kiírja: 1
MyChildClass mcc;
mcc.PrintValue(); // kiírja: 2
mcc.PrintBaseValue(); // kiírja: 1
```


Öröklődés

A konstruktor öröklődése

- A konstruktor automatikusan öröklődik
 - a paraméteres nélküli konstruktor automatikusan meghívódik amikor a leszármazottból létrehozunk egy példányt
 - elsőként az ős konstruktora hívódik meg, aztán a leszármazottaké lefelé haladó sorrendben
 - lehetőségünk van az ős konstruktorának explicit meghívására is `<osztálynév> <konstruktor> : <ős konstruktornév>(<átadott paraméterek>)` formában
 - paraméteres konstruktorokra csak az explicit hívás használható

Öröklődés

A konstruktor öröklődése

- Pl.:

```
class MyBaseClass {  
public:
```

```
    MyBaseClass() { cout << "1" << endl; }  
};
```

```
class MyChildClass : public MyBaseClass {  
public:
```

```
    MyChildClass() { cout << "2" << endl; }  
};
```

...

```
MyChildClass mcc; // konstruktor hívás  
// eredmény: 1 2
```

Öröklődés

Példa

Feladat: Készítsünk egy programot, amelyben téglalapokat és köröket tudunk a képernyőre rajzolni, amelyek három színben (piros, fehér, zöld) váltakoznak. A színeket utólag le tudjuk cserélni (új véletlenszerű színre), és a teljes rajzot el tudjuk tolni egy vektorral valamilyen irányba.

- javítsunk a korábbi megoldáson azzal, hogy az ős (**Shape**) konstruktorára bizzuk a mezők (`_point`, `_color`) inicializálását
- a leszármazott osztályok konstruktorai csak meghívják az ős konstruktorát, és tovább adják a kapott paramétert

Öröklődés

Példa

Tervezés:

