



Eötvös Loránd Tudományegyetem
Informatikai Kar

Alkalmazott modul: Programozás

14. fejezet

Objektumorientált programozás: osztályszintű adatkezelés

Giachetta Roberto

A jegyzet az ELTE Informatikai Karának 2015. évi
Jegyzetpályázatának támogatásával készült

Osztályszintű adatkezelés

Statikus változók

- Változók tekintetében megkülönböztetjük:
 - *globális változókat*: a teljes programban jelen vannak, és mindig, bárhol elérhetőek
 - *lokális változókat*: egy adott blokkon belül jönnek létre, és a létrehozás pontjától a blokk végéig élnek
- Lehetőségünk van olyan változókat létrehozni, amelyek láthatóság tekintetében lokálisak, de a deklaráció pontjától a program végig életben vannak, és az értékük újradeklarálás hatására sem veszik el, ezek az úgynevezett *statikus változók*
 - félúton vannak a globális és lokális változók között
 - létrehozásuk a **static** kulcsszóval történik

Osztályszintű adatkezelés

Statikus változók

- Pl.:

```
void SomeFunction() {
    int value = 0;
    // lokális változó létrehozása
    value++; // módosítása
    cout << value << " ";
}

int main() {
    for (int i = 0; i < 10; i++)
        SomeFunction();
    return 0;
}

// eredménye:
// 1 1 1 1 1 1 1 1 1 1
```

Osztályszintű adatkezelés

Statikus változók

- Pl.:

```
void SomeFunction() {
    static int value = 0;
    // statikus változó létrehozása
    value++; // módosítása
    cout << value << " ";
}

int main() {
    for (int i = 0; i < 10; i++)
        SomeFunction();
    return 0;
}

// eredménye:
// 1 2 3 4 5 6 7 8 9 10
```

Osztályszintű adatkezelés

Statikus mezők

- Statikus értékeket használhatunk osztály tagjaiként is, ezeket *osztályszintű*, vagy *statikus mezők*nek nevezzük
 - olyan adatok, amelyek eléréséhez nem kell az osztályt példányosítani, elég hivatkozni rá az osztályon keresztül
 - a hivatkozás a `::` (scope) operátorral történik
`<osztálynév>::<változónév>` formában (de objektumon keresztül is elérjük őket)
 - a használat előtt külön definiálnunk kell, akkor is, ha példányosítjuk az osztályt, ugyanis a statikus változó nem fog vele példányosulni
- A statikus mezőket az osztálydiagramban aláhúzással jelöljük

Osztályszintű adatkezelés

Statikus mezők

- Pl.:

```
class SomeClass{  
public:  
    int value; // mező  
    static int sValue; // statikus mező  
};
```

```
int SomeClass::sValue = 0;  
    // statikus mező definíciója
```

Osztályszintű adatkezelés

Statikus mezők

- Pl.:

```
int main() {
    cout << SomeClass::sValue << endl; // kiírja: 0
    SomeClass a, b;
    a.sValue = 1;
    // hozzáférünk az objektumon keresztül is
    a.value = 1;
    // az egyszerű mezőhöz csak az objektumon
    // keresztül férünk hozzá
    cout << b.sValue << endl; // kiírja: 1
    b.sValue = 3;
    cout << a.sValue << endl; // kiírja: 3
    return 0;
}
```

Osztályszintű adatkezelés

Statikus metódusok

- Ahogy lehetőségünk van osztályszintű mezők létrehozására, úgy *osztályszintű metódusok* létrehozására is
 - olyan metódusok, amelyek közvetlenül az osztályból meghívhatóak (tehát nem kell példányosítanunk az osztályt)
 - a metódust szintén a **static** kulcsszóval jelöljük meg
 - a metódus futtatható osztályon keresztül is a **::** operátorral, valamint objektumon keresztül is
 - cserébe nem érjük el az aktuális objektumot (nincs **this**), és csak az osztályszintű mezők láthatóak benne
- A statikus metódusokat az osztálydiagramban aláhúzással jelöljük

Osztályszintű adatkezelés

Statikus metódusok

- Pl.:

```
class SomeClass {
public:
    int value; // mező
    static int sValue; // statikus mező

    static void sMethod(int v) {
        // statikus metódus
        sValue = v; // a statikus mezőhöz hozzáfér
    }
};

int SomeClass::sValue;
// statikus mező definíciója
```

Osztályszintű adatkezelés

Statikus metódusok

- Pl.:

```
int main() {
    SomeClass::sMethod(1);
    // statikus metódus meghívása
    cout << SomeClass::sValue << endl; // kiírja: 1

    SomeClass a, b;
    a.sMethod(3);
    // objektumon keresztül is meghívhatjuk
    cout << b.sValue << endl; // kiírja: 3
    return 0;
}
```

Osztályszintű adatkezelés

Használata

- A statikus mezőket és metódusokat azért célszerű használni, mert:
 - elkerülhető a globális változók alkalmazása
 - viselkedése megegyezik a globális változóéval
 - az érték nem lesz érvényes a teljes programban, csak abban a tárgykörben, amelyben szeretnénk használni
 - elkerülhető az osztály példányosítása
 - a statikus tag elérhető példányosítás nélkül
 - közös pontot biztosít minden osztálypéldány számára
 - az objektumok könnyen kommunikálhatnak, más, ugyanolyan osztályú objektumokkal

Osztályszintű adatkezelés

Példa

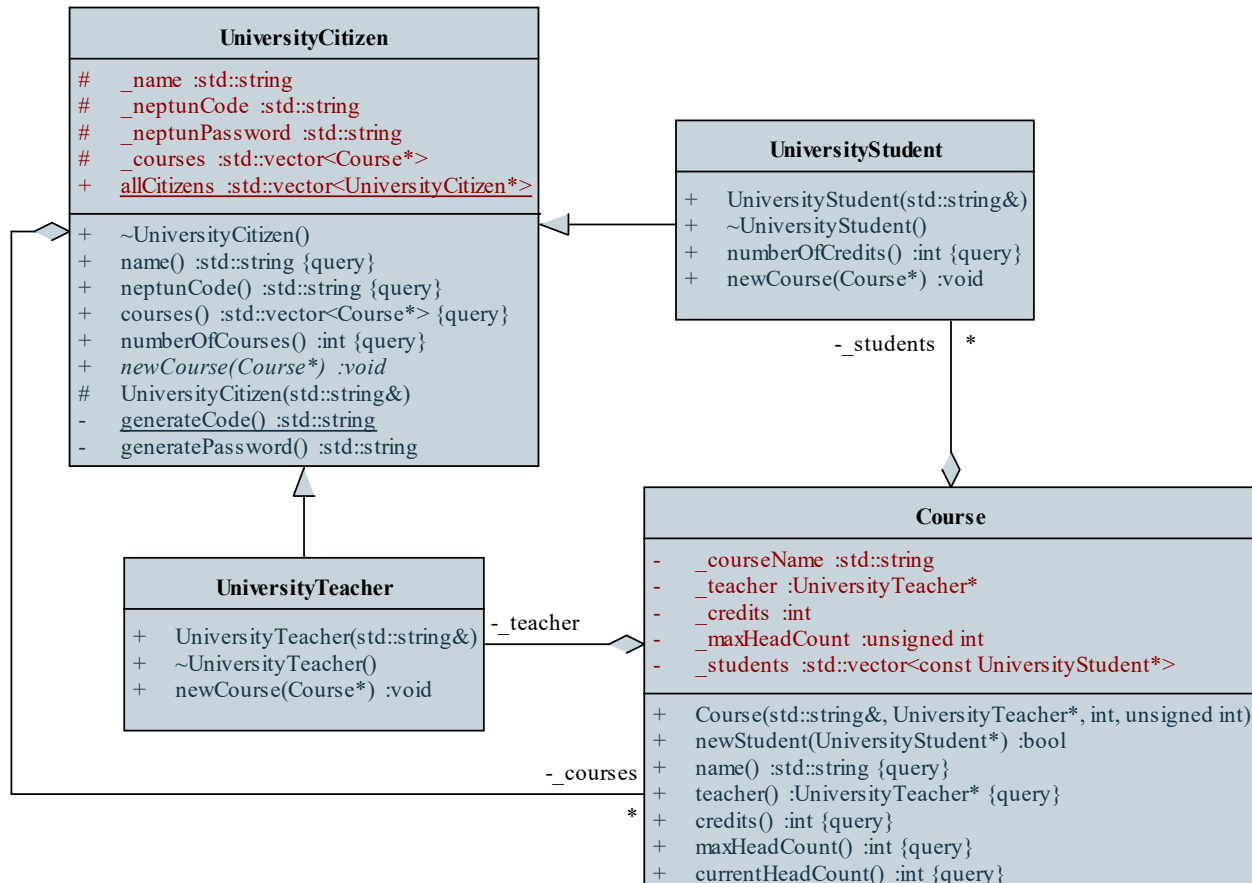
Feladat: Javítsuk az egyetemi modellünket úgy, hogy két egyetemi polgár ne kaphassa ugyanazt a Neptun kódot.

- ehhez létrehozunk egy osztályszintű vektort (**`allCitizens`**), amelyben minden polgárt (pontosabban a mutatóját) behelyezünk a létrehozásakor
- a vektort felhasználjuk arra, hogy minden Neptun kód generálásakor ellenőrizzük, nem-e rendelkezik valaki már a megadott kóddal, ha igen, akkor újat generálunk
- ezt a vektort felhasználhatjuk az összes polgár kiíratásához
- figyeljünk arra, hogy polgár törlésekor ki kell őt a vektorból is törölni

Osztályszintű adatkezelés

Példa

Tervezés:



Osztályszintű adatkezelés

Példa

Megoldás (universitystudent.cpp):

...

```
UniversityCitizen::UniversityCitizen(const
    string& n) : _name(n) {
    _neptunCode = generateCode();
    _neptunPassword = generatePassword();

    allCitizens.push_back(this);
    // az összes polgár közé felvesszük a most
    // létrehozottat is
}
```

...

Osztályszintű adatkezelés

Példa

Megoldás (main.cpp):

```
...
for (int i = 0;
     i < UniversityCitizen::allCitizens.size();
     i++) {
    // eleve el vannak tárolva az egyetem polgárai
    // egy osztályszintű listában
    if (dynamic_cast<UniversityTeacher*>(
        UniversityCitizen::allCitizens[i]))
        // ha oktató
        cout << UniversityCitizen::allCitizens[i]
             ->name() << " (oktató), NEPTUN: " <<
             UniversityCitizen::allCitizens[i]
             ->neptunCode() << endl;
}
...
```