5. LABOR

UML, GYAKORLÁS OSZTÁLYOKKAL, DINAMIKUS OSZTÁLYOK

Ezen a laboron az eddig tanultakat lehet egy összetettebb feladaton gyakorolni. A 2-4. feladatot közösen meg fogjátok beszélni. Az 5. feladatot önállóan készítsétek el és teszteljétek.

Ezen az órán üres projektből induljunk ki.

Feladatok

1. UML ismétlés

Hallgasd meg és jegyzeteld le a laborvezetőd ismétlését az UML alapú modellezésről. Jegyezd meg a jelöléseket, ezeket később továbbiakkal egészítjük ki. A zárthelyin egyes feladatokat hasonló módon kell majd modellezned.

2. OO tervezés UML diagrammal

Egy Mindenízű Gumimacikkal kereskedő MLM (multi-level marketing) cég partneradatbázisát karbantartó szoftver tervezési feladatát kaptuk meg. A cég partnerei magánszemélyek lehetnek (Person). A magánszemélynek van vezetékneve (lastName), keresztneve (firstName), adóazonosító jele (9 karakter, taxId), és egy fixen 2000 karakteres egyedi titkosítási kulcsa (privateKey), valamint rámutathat egy másik magánszemélyre, aki őt beajánlotta partnernek (contact). Először még ne dinamikus tagokkal tervezzünk, hanem fix karaktertömbökkel (pl. 19+1 karakteres nevek).

A titkos kulcsot egy függvény (generatePrivateKey) segítségével bármikor (újra) előállíthatjuk.

Az UML tervezés során gondoljuk végig, mi a privát, mi a publikus tagváltozó (adatrejtés elvét figyelembe véve). Tervezzünk a tagváltozókhoz getter/setter függvényeket is a diagramunkra! A konstruktorokkal még ne törődjünk.

Az alap osztálydiagram elkészítése után vegyünk fel egy függvényt, ami kiírja a magányszemély adatait (nevét és adószámát), pl. nyomtat vagy print névvel.

3. Osztály készítése

A fenti tervet írjuk le C++ nyelven osztályokkal.

A mezők kapcsán minimiális **hibakezelés** is legyen, pl. csak akkor állítjuk be az adóazonosítót, ha pontosan 9 karakter a megadott. Említhetjük, hogy később lesz erre rendes módszerünk is (Exception).

Készítsünk egy függvényt, ami véletlenszerűen generálja a 2000 karakterből álló titkos kulcsot

setTaxId: **strcpy**

- Miért kell, mi van, ha nem használjuk?
  - Terminálo null karaktert is másolja, nem nekünk kell figyelni rá.
  - Nem állíthatjuk át csak a pointert, deep copy kell (miért?)
- Mi van, ha hosszabb a paraméterként megadott string, mint a tagváltozónak lefoglalt terület?
  - **strcpy**: Overflow keletkezik (akkor is, ha pont annyi karaktert adunk meg, ahány karakternek hely van lefoglalva, mert null-terminál karakter hiányzik). Jobb compilerek figyelmeztetnek.
  - **strcpy_s**: Megadható a maximális mérete a cél buffernek, ha ennél több karakterből álló karaktertömböt másolnánk bele, hibát kapunk. Futási időben ellenőriz, viszont nem szabványos.

Írjunk kódot is, amivel kipróbáljuk.

4. Konstruktorok

Készítsük el az osztály konstruktorait.

- Default konstruktort nem szeretnénk.
- Készítsünk konstruktort, amely csak a nevet kéri be.
- Készítsünk konstruktort, amely a nevet és adóazonosítót kéri be.
- Gondoljuk közösen végig, hogy az OO elvekre a tervezésnél hogyan tudunk figyelni. Jó-e jelenleg a tervünk? Mi sérülhet és hol?
- A megoldások lehetnek kreatívak, a minta megoldást tovább kiegészítve (pl. titkos kulcs generálás konstrukció során automatikusan megtörténik stb).
- Destruktor kérdésköre: ki kell-e törölni a beajánló személyt a destruktorban? Miért?

5. Dinamikus tagváltozók

Írjuk át úgy a megoldást, hogy a privát kulcs tagváltozó dinamikus legyen, a heap memóriában tároljuk, hiszen nagy a mérete. Ezért a tagváltozóért, az általa foglalt memória területért a Person osztály lesz felelős, itt kompozícióról lesz szó. Miket kell módosítani a tagváltozókon kívül? (Gondold végig az órai ökölszabályokat a Person lemásolásával, megszűnésével kapcsolatban). Használhatod segítségképpen az órai példát is.

Teszteljük is az eredményt Person példány létrehozásával, lemásolásával.