A programozás alapjai 2.

Hivatalos segédlet a harmadik laborhoz.

Tartalomjegyzék

Mik azok az osztályok és objektumok?

Osztály (class): egy új típus C++-ban

- függvények és tulajdonságok gyűjteménye
- a tagfüggvények viselkedést definiálnak
- a tagváltozók állapotot írnak le
    - tagváltozó típusa lehet egy másik osztály → tartalmazás kapcsolat
    - pl. számítógépben van CPU, RAM → Computer osztályban lesz egy Cpu és egy vagy több Ram típusú tagváltozó
    - pl. a szobának része az ajtó, ablak stb.
- egyszer kell megírni

Objektum (object): egy osztály egy példánya

- osztály definiálja a struktúrát
- különböző objektumok tagváltozóinak értékei különbözők
- annyiszor van létrehozva, ahányszor szeretnénk (memória azért gátat szab ennek, de nagyon sok lehet belőle)

Class vs Struct

	Class	Struct
Használható C-ben?	nem	igen
Használható C++-ban?	igen	igen
Alapértelmezett láthatóság	private	public

Objektum-orientált tervezési elvek 1.

- **Absztrakció (abstraction)**
    - a feladat szempontjából lényegtelen részleteket figyelmen kívül hagyjuk
        - pl. alkalmazottak nyilvántartásakor nem szükséges tudni (általában) a személyek kisujjainak hosszát
    - a világban létező dolgok leképezhetők a program objektumaira
- **Osztályozás (classification)**
    - az ugyanolyan viselkedéssel és tulajdonságokkal rendelkező dolgokat csoportosítjuk
    - ezeknek a viselkedése és tulajdonságai ugyanazzal az osztállyal írhatók le
- **Beágyazás (encapsulation)**
    - az összetartozó adatokat és a rajtuk értelmezett műveleteket egy osztályba helyezzük, így együtt egy egységet alkotnak
- **Adatrejtés (information/data hiding)**

- egy osztály nem enged közvetlen hozzáférést az tagváltozóihoz
- csak függvényeken keresztül érhetők el
- minden tagváltozónak privátnak kell lennie
- így elkerülhető az objektum inkonzisztens állapotba kerülése (csak olyan változtatást hajtunk végre a függvényben, ami engedélyezett és értelmes)

Konstruktor

Az osztályok speciális tagfüggvénye, ami akkor hívódik meg, amikor az osztályból példányosítunk egy új objektumot. Neve megegyezik az osztály nevével és nincs semmilyen visszatérési értéke (még void sem). Általánosan az objektum kezdőértékeinek beállítására használható.

Egy osztály konstruktora túlterhelhető, többféle paraméterezésűt létre lehet hozni (pl. ha a Rectangle osztályt két paraméterrel hívjuk, akkor téglalapként viselkedik, ha csak eggyel, akkor a magassága és szélessége is a megadott paraméterrel lesz egyenlő, és négyzetet reprezentál). Alapértelmezett értékek beállíthatók a paramétereinek a korábban tanult szabályok alapján.

Explicit nem lehet meghívni, az objektum létrehozásakor automatikusan hívódik meg a megfelelő paraméterezésű konstruktor, és egyszer fut le.

Alapértelmezett (default) konstruktor

Ha a konstruktornak nincs egy paramétere sem, **default konstruktor**-nak nevezzük.
Ha mi magunk nem írunk egyetlen konstruktort sem, alapértelmezetten lesz egy default konstruktora az osztályunknak. Amint írunk bármilyen másik konstruktor (például egy **másoló konstruktort**), de default konstruktort nem, akkor nem fog létrejönni automatikusan default konstruktor.
Megjegyzendő, hogyha nem definiálunk külön default konstruktort, de van olyan konstruktor, amelyben minden paraméternek van default értéke (tehát meghívható akár paraméter nélkül), akkor nem kell külön default konstruktort írni.

Másoló konstruktor

Paraméterként olyan típusú objektumot vesz át, mint amelyik osztály konstruktoráról van szó. Ekkor a paraméterül kapott objektum tagváltozóinak értékét le kell másolni az új objektum tagváltozóiba. Fontos, hogy mindenhol **deep copy**-t kell készíteni. Ez azt jelenti, hogy ha van pl. egy pointer tagváltozónk, akkor az új objektumban nem egyszerűen létrehozunk egy pointert, amivel rámutatunk a már meglévő memóriaterületre. A helyes megoldás, ha lefoglalunk egy ugyanakkora memóriaterületet, mint amekkora az eredetiben van, és minden értéket átmásolunk az új memóriaterületre is. Az új objektumban lévő mutató erre az új memóriaterületre fog mutatni.

Destruktor

A destruktor egy olyan speciális tagfüggvény, ami akkor hívódik meg automatikusan, amikor az adott objektum életciklusa végetér:

- *delete*-t hívnak rajta
- program vagy scope (pl. függvény) terminál
- kezeletlen exception (kivétel) keletkezik
- közvetlenül is meghívható, de ilyet nem szoktak (helyette delete)

Nem írunk destruktort, a compiler mindig készít egyet inline public láthatósággal.

Ha az

- osztályon belül egy tagváltozónak new-val foglaltok típusos memóriaterületet,
- fájlt vagy hálózati kapcsolatot nyittok meg (ezek drága erőforrások, hiszen más processek várakozhatnak rájuk → ha lehetőség van rá, még a destruktor lefutása előtt fel kell szabadítani őket!)

akkor a **destruktorban mindig fel kell szabadítani**. A fentebbi bekezdésben említett, a compiler által automatikusan létrehozott destruktor ezt nem végzi el, mindig nekünk kell megírnunk.

Láthatóság

Az osztályok tagváltozói és tagfüggvényei rendelkeznek ún. láthatósággal. Ez azt adja meg, hogy az adott tagváltozót ki érheti el közvetlenül és az adott tagfüggvényt ki hívhatja meg. A két alapvető láthatóság a következő:

public

Mindenhonnan hozzáférhető a programban közvetlenül, osztályon kívüli függvényből is. Ez azt jelenti, hogy nem kell függvényt hívni ahhoz, hogy megkapjuk vagy megváltoztassuk az értékét egy tagváltozónak.

private

Osztályon kívülről nem érhető el, nem is látható. Ez az alapértelmezett láthatóság az osztályokban. Az osztály függvényeiből viszont elérhető.

- **getter**: Olyan metódus, amely visszaadja egy privát láthatóságú tagváltozó értékét. A metódus maga publikus. Elnevezése "get" prefix-szel kezdődik, és a tagváltozó nevével folytatódik.
  pl. privát "age" tagváltozó getter metódusának fejléce: int getAge()
- **setter**: Olyan metódus, amely megváltoztatja egy privát láthatóságú tagváltozó értékét. A metódus maga publikus. Elnevezése "set" prefix-szel kezdődik, és a tagváltozó nevével folytatódik. Ebben a függvényben lehetőségünk van a paraméterül átvett érték ellenőrzésére. Ha nem megfelelő értéket adott meg a hívó, nem végezzük el a módosítást. Így biztosítható pl. hogy az e-mail címnek e-mail cím formátuma legyen.
  pl. privát "age" tagváltozó setter metódusának fejléce: void setAge(int newAge) → ekkor, ha a "newAge" értéke negatív, nem végezzük el a változtatást, mert az nem érvényes kor.

Léteznek más láthatóságok is, melyekről a későbbiekben esik majd szó.

A *this* pointer

Osztályok tagfüggvényein belül érhető el, 0. paraméterként automatikusan megkapja. A tagfüggvényen belül arra az objektumra mutat, melyen a függvényt meghívták. Ezen keresztül elérhetők az aktuális objektum tagváltozói és tagfüggvényei (privát láthatóságúak is).

C++ elnevezési konvenciók

- **típus**: UpperCamelCase
  - szavak között nincs elválasztó karakter
  - az minden kezdőbetű nagybetű

  - főnév
  - pl. BookStore
- **függvény**: lowerCamelCase
  - ige
  - szavak között nincs elválasztó karakter
  - első betű kisbetű
  - minden más kezdőbetű nagybetű
  - pl. getOldestBook()
- **változó/tagváltozó**: lowerCamelCase (ld. függvény elnevezése)
  - pl. firstName
- **konstans**: UPPER_CASE
  - csupa nagybetű
  - szavak aláhúzással elválasztva
  - pl. MAX_TABLE_SIZE

Clean Code 1. - Általános

- **Miért tartsuk be** ezeket és a későbbiekben bemutatott elveket (azon felül, hogy a házi feladatban elvárt ezeknek az elveknek a betartása)?
  - könnyebb, gyorsabb megérteni a programot (készítő és más személy által is), így **időt takarítasz meg** magadnak és megkönnyíted a saját életed (a befektetett odafigyelés 10-szeresen térül meg a későbbiekben)
  - **könnyebb változtatni** valamit a programon (és egy program soha sincs kész, mindig változik)
  - **könnyebb** lesz **kijavítani a bug-okat**
- Nagyon hosszú sorokat **tördeld** (max. 80 karakter legyen egy sor)
- Használj **behúzásokat**, ne minden egy oszlopban kezdődjön
- Használj **értelmes nevek**et, amik felfedik a változó/függvény/osztály célját
  - kereshető
  - öndokumentáló
- **Különböző célokra különböző változók**at használj
- Don’t Repeat Yourself (**DRY**): Ne ismételd magad!
  - a duplikáció minden probléma gyökere
  - ha valamin változtatni kell később, mindenhol változtatni kell (pl. bug javítás)
  - valószínű, hogy valahol elfelejted megváltoztatni, és nehéz lesz kideríteni, miért nem működik mégsem a kód

Clean Code 2. - Függvények

- Egy függvény **max 20 sor** legyen
  - különben bontsd fel több függvényre
- **Egy függvény egy felelősség**et valósít meg (ha több szekcióra osztható, akkor valószínűleg több dolgot csinál)
- Függvényeknek **max 3 paramétere** legyen (különben nehéz tesztelni minden kombinációt)
- **Logikai értéket átadni függvénynek csúnya**, valószínűleg több dolgot csinál a függvény, ezért kerüljük messziről
- Függvényeknek **ne legyen váratlan mellékhatás**a
  - pl. tömb elemeit kiíró függvény ne változtassa meg a tömb elemeit

- Egy függvény **vagy változtat valamin vagy információt szolgáltat** róla, egyszerre a kettőt nem csinálja
- Védd magad: **ellenőrizd** a bemeneti paramétereket, ne feltételezd, hogy jók

Clean Code 3. - Kommentek

Milyet használjunk?

- **Informatív** komment: bonyolult kifejezés magyarázata
- **Szándék** magyarázata: miért csinálja a kód azt, amit
  - főleg ha érdekes, furcsa megoldásról van szó vagy bugfix
- **Figyelmeztetés** a következményekről
  - pl. sok ideig tart a futtatás, készüljön fel rá, aki használja
- **Pontosítás**: bonyolult szintaxis, mit jelent egy bizonyos visszatérési érték
  - pl. compare függvény visszatérési értékei jelentésének magyarázata -1, 0, 1
- **TODO** komment: később még át kell írni, vagy még nincs kész az, amit használna
- Nem egyértelmű rész **fontosságának** jelölése
  - pl. bináris keresés előtt muszáj rendezni az adatokat
- Függvény paramétereinek elvárt **értéktartománya**, visszaadott értékek tartománya, milyen hiba léphet fel
- Mik az **alapértelmezett értékek**

Milyet NE használjunk?

- **Redundáns** komment, ami könnyen kiolvasható a kódból
  - for(i = 0; i < 10; i++){} // ciklus 0-tól 9-ig
- Komment jól elnevezett változó/függvény helyett
  - int asd = 12; // hónapok száma egy évben HELYETT int monthsInYear = 12;
- **Kódrészlet** kikommentezve
- **Túl hosszú** komment
- **Nem teljes** komment: egyes részek meg vannak magyarázva, mások nincsenek

Clean Code 4. - Osztályok

- **adatrejtés** elve miatt minden **tagváltozó** legyen **private**, amihez biztosíts getter metódusokat, ha szükséges, akkor setter-t is
- egy osztálynak **egy felelőssége** legyen
- bővítés során régi kód ne változzon
- **ne** használj olyan osztályokat, amiknek csak **tagváltozóik** és **getter/setter** metódusaik vannak
  - nincs viselkedése
- a **kapcsolódó tagváltozók és viselkedés** egy osztályban legyen
- **kevés publikus függvénye** legyen az osztálynak
  - különben nehéz megtalálni azt, amire szükség van
  - ugyanannak a dolognak az elvégzése csak egyféleképp legyen végrehajtható

- implementálj **standard függvényeket**
  - egyenlőség-vizsgálat
  - kiírás
  - deep copy