Automatizálási és
Alkalmazott
Informatikai Tanszék

PROGRAMOZÁS ALAPJAI 2. (VIAUAA01)

4. LABOR IMSC FELADAT

DINAMIKUS TAGVÁLTOZÓK, DINAMIKUS OSZTÁLYOK

KOVÁCS LEVENTE (F5UHYT)

6. Áttekintés: Person osztály

1. A dinamikusan foglalt tagváltozóknál mindig fontos, hogy megfelelően kezeljük a másolást és a felszabadítást. Általánosságban igaz (ahogy ez a *Person* osztályban is van), ha egy osztály tartalmaz heapről allokált memóriát, akkor másoláskor az új objektumnak újonnan foglalt terület kell, megszűnéskor pedig fel kell szabadítani a foglalt területeket.

2. Egy már létező *Person* objektumot úgy tudunk másolni, hogy létrehozunk egy új objektumot, majd értékül adjuk neki a lemásolandót. Ilyenkor az ún. *copy constructor*, vagy másnéven másoló konstruktor hívódik meg. A másoló konstruktor egy olyan speciális függvény, mely egyetlen paramétere egy ugyanolyan osztályú objektumra egy referencia. A *Person* implementációjában ez csupán annyit jelent, hogy a *name* tagváltozónak új terület lesz foglalva, majd a sztring átmásolásra kerül.
3. A konstruktor egy objektum létrehozásakor jön létre. Ennek az a feladata, hogy akár alapértelmezett, akár megadott adatokkal inicializálja a létrejövő objektumot. A destruktor az objektum megszűnésekor hívódik meg, a feladata gyakran a dinamikusan foglalt memóriaterületek felszabadítása.

Az ábrán látható konzol log a forrásfájlokban található utasítások szerint soronként lebontva:

1. Létrejön dinamikusan foglalt p1: `Person* p1 = new Person();`
2. Létrejön p2 (stacken): `Person p2("Aladar", 13);`
3. p2 nevét a paraméteres konstruktor beállítja: `this->setName(name);`
4. Meghívódik a copy constructor: `Person p3(p2);`
5. Copy constructor meghívja a *setName*-en belül *getName*-t: `this->setName(p.getName());`
6. Meghívódik a fenti *setName()*
7. Copy constructor beállítja az új példány korát: `this->age = p.getAge();`
8. p1 nevét Bélára állítjuk: `p1->setName("Bela");`
9. p1 korát 15-re állítjuk: `(*p1).setAge(15);`
10. Dinamikusan foglalt p1 (Béla, 15) megszűnik: `delete p1;`
11. p2 (eredeti Aladar) megszűnik: `return 0;`
12. p3 (p2 másolata) megszűnik: `return 0;`

```
1.  Default konstruktor lefut
2.  Parameteres konstruktor lefut
3.  setName lefut
4.  Copy konstruktor lefut
5.  Aladar  getName lefut
6.  setName lefut
7.  Aladar  getAge lefut
8.  setName lefut
9.  Bela    setAge lefut
10. Bela    destruktor lefut
11. Aladar  destruktor lefut
12. Aladar  destruktor lefut
```

Konzol log

7. VendingMachine osztály

1. A *VendingMachine* osztályon belül a *drinks* egy C++ sztringekből álló tömb. A *std::string* típus a *std* névtéren belül egy meglévő és használható implementációt biztosít a sima C „sztringekkel” szemben. Ez egyszerűvé teszi a C++ sztringek kezelését, egyszerűsített működést biztosítanak. A C-vel való visszafele kompatibilitást a c_str() függvény biztosítja. Ez a függvény egy objektumon keresztül meghívva visszad egy pointert egy NULL-terminált karaktertömbre. Pl.:

```
const char *c_Cola = Cola.c_str();
```

2. A konkrét implementációban a hatékonyságbeli problémát az jelenti, hogy minden alkalommal, amikor változtatjuk a *drinks* méretét (*refill(), buy(), removeOne()*) akkor új tömböt kell foglalni és az összes szükséges elemet átmásolni. Erre egyszerű megoldást jelenthet egy más adatstruktúra (pl. láncolt lista) választása. (Itt jön elő igazán az objektumorientáltság és az osztályok előnye: a *VendingMachine* osztályt a public interfacen keresztül használó kódhoz hozzá se kell nyúlni az adat tárolási módjának teljes megváltozása ellenére sem.)