6. LABOR

NÉVTEREK, KONSTANS ÉS STATIKUS DEKLARÁTOROK

Általános információk

1 iMSc pont jár a 3. és 4. feladat együttes teljesítéséért. Az 4. feladatnál elég egy rövid kódrészlet, amiben bemutatod, hogy tényleg csak egyszer lehet példányosítani a Logger osztályt.

Kötelező feladatok

1. Computer osztály

Adott egy kódrészlet, amellyel számítógépeket (Computer) modellezünk. Minden Computer példány egyedi, amit a gyártói azonosítója (id) mutat. Ez az id egy 0-ról induló, folyamatosan növekvő szám lesz: minden újonnan létrehozott példány eggyel nagyobb azonosítót kap. Ebben fog segíteni a nextId tagváltozó is. A számítógépek processzor sebességét is eltároljuk, amihez egy lekérdező függvényt adunk.

Egészítsd ki a megadott *Computer* solution állományait a következőknek megfelelően:

- A *computer.h*-ban pótold a hiányzó *static*, *friend* és *const* kulcsszavakat! (Mi legyen konstans, statikus tagváltozó? Melyik tagfüggvénynek kell konstansnak lennie, és miért? Van-e olyan függvény paraméter, aminek konstansnak kell lennie? Mi kell ahhoz, hogy a friendlyPrint függvény hozzáférjen a Computer összes privát tagváltozójához is getterek nélkül? Hogyan lehetne egyébként ezt szebb, objektumorientált elveknek jobban megfelelő kóddal elérni?)
- A *computer.h*-nak megfelelően írd meg a *computer.cpp* fájlt.
- Ügyelj, hogy az egyes másolatoknak ne legyen azonos az id-je (használd a nextId-t ebben azesetben is!)
- Ügyelj arra, hogy a mellékelt *computerTest.cpp* helyesen lefusson.

2. Termékek nyilvántartása statikus és konstans elemekkel

Egészítsd ki a megadott *Product* solution állományait a következőknek megfelelően:

Készíts egy **Product osztályt**, amely egy boltban található terméket reprezentál. Az osztálynak képesnek kell lennie nyilvántartani a termék azonosítóját, nevét és árát, valamint statikusan követni az összes létrehozott termék számát és egy előre meghatározott maximális árhatárt. (Fontos: a Product csak egyetlen terméket tárol, nem a teljes készletet!)

Elvárások a Product osztállyal kapcsolatban:

- **Statikus tagváltozók:**
  - totalProducts: nyilvántartja, hány terméket hoztunk létre.
  - MAX_PRICE: egy **statikus konstans** változó, amely a maximális lehetséges árat jelenti (pl. 5000 Ft). Ezt a konstrukció során sem szabad túllépni!
- **Konstans tagváltozók:**

  - id: minden termék egyedi azonosítója, ami nem változhat létrehozás után.
  - name: a termék neve (szintén nem változik)
  - price: a termék ára.
- **Statikus függvények:**
  - getTotalProducts(): visszaadja az összes termék számát.
  - getMaxPrice(): visszaadja a maximális megengedett árat.
- **További tagfüggvények:**
  - getId(), getName(), getPrice(): visszaadják a megfelelő értékeket.
  - printDetails(): kiírja a termék adatait, **de csak a gettereket használhatja!**
- **A printDetails() függvényt mindenképp szeretnénk konstans objektumon is meghívni.**
- **A main() függvényben csak konstans Product példányokat hozzunk létre.**

3. Logger singleton osztály

Egy általad tervezett beágyazott rendszerre írtál egy programot C++-ban és szeretnél globális eseménynaplózást megvalósítani (hálózati kapcsolat megszűnt/helyreállt, abnormálisan magas hőmérséklet a processzorban (junction temperature). Úgy döntöttél, hogy a jól bevált Singleton tervezési mintát alkalmazod a probléma megoldására.

Készíts egy Logger singleton osztályt, ami legyen képes az alábbiakra:

- A megadott *loggingTest.cpp* állománynak megfelelően írt meg a *logging.h* és a *logging.cpp* fájlokat.
- Ügyelj a helyes névtér használatra.
- Használj enumerációt ott, ahol kell.
- Biztosítsd a következő naplózási szinteket:
  - DEBUG: minden eseményt naplóz
  - INFO: sikeres/sikertelen végrehajtást naplóz
  - WARN: a jövőre nézve veszélyes eseményeket naplóz
  - ERROR: hibaeseményt naplóz
- Legyen beállítható a minimális naplózási szint
  - csak egy bizonyos szint és az afeletti szintek eseményeit naplózza
  - használd ezt a sorrendet: DEBUG, INFO, WARN, ERROR

Segítségek:

- A Logger osztály nem tárol naplózási üzeneteket, csak magáért a naplózásért, kiíratásért felelős.
- A setDefaultLogLevel függvény arra való, hogy meghatározzuk, melyik naplózási szint (és felette) vagyunk kíváncsiak a napló üzenetekre. Ha ez pl. ERROR, ebben az esetben a DEBUG infókat nem akarjuk jelenleg kiíratni (erre a log függvényben kell figyelnünk). Így mindig csak az olyan fontosságú üzeneteket fogjuk látni, amikre az adott futás során szükségünk van.

4. Biztos, hogy singleton?

Győződj meg róla, hogy akárhányszor hívod meg a Logger::getInstance() függvényt, mindig ugyanazt a példányt kapod vissza.