PROGRAMOZÁS ALAPJAI 2. (VIAUAA01)

6. LABOR IMSC FELADAT

NÉVTEREK, KONSTANS ÉS STATIKUS DEKLARÁTOROK

KOVÁCS LEVENTE (F5UHYT)

4. Logger singleton osztály

A singleton tervezési minta azt jelenti, hogy egy adott osztályból csak egy példány hozható létre. A példányosítás korlátozásának hasznosságára tökéletes példa a feladatban mutatott logging feladatot ellátó osztályok.

Egy singleton osztály egyszerűen implementálható:

1. Az osztály konstruktora és destruktora értelemszerűen privát kell hogy legyen a példányosítás megakadályozásához
2. Ilyenkor viszont szükségünk van egy függvényre, ami létrehozza azt az egy példányt amire szükségünk van és vigyáz arra, hogy több példány ne jöhessen létre. Ezt legegyszerűbben egy statikus függvénnyel tehetjük meg, amiben létrehozunk egy statikus példányt az osztályunkból. Itt a statikus deklarálás biztosítja azt, hogy a függvényünkben (tipikusan *getInstance()*) csak egyszer inicializálódhat egyetlen darab példány az osztályunkból.
3. A másolást megelőzendő fontos kiiktatni a másoló konstruktor lefutásának összes lehetőségét. Ezt maga a copy constructor és az értékadás (’=’) operátor törlésével tehetjük meg.

```
Logger(const Logger &) = delete;
void operator=(const Logger &) = delete;
```

A feladat többi részének a megvalósítása triviális. Az implementációmban a log egy fileba készül, aminek az elnevezése a time standard C függvénykönyvtár segítségével a pontos időhöz kötött. A singleton log instance inicializálásakor létrejön egy *„[ÉV]-[HÓNAP]-[NAP]_[ÓRA]-[PERC]-[MÁSODPERC]_log.txt”* szöveges fájl és a logolás kezdetének időpontjával belekerül egy *„Log started”* üzenet. Innentől kezdve az egyetlen *Logger* példányon keresztül hívott *log()* függvény ebbe a fájlba ír egy adott:

„[ÓRA]:[PERC]:[MÁSODPERC]: [LOG ÜZENET]”

formátumú log sort.

A logolási szintek implementációjához 4 szint van specifikálva: *DEBUG, INFO, WARN, ERROR*. Egy log csak akkor kerül bejegyzésre, ha a *log()* hívásánál megadott szint a *setDefaultLogLevel()* által specifikált szint alatt van. A konstruktor alapesetben *ERROR*-ra állítja a logolási szintet.

5. Biztos, hogy singleton?

A fentebb említett példányosítás kiküszöbölésének tesztelése az alábbi módon történik:

```
1.      // If logging::Logger is singleton, this is a reference to the same instance
2.      Logger &loggerSingletonEEzACsoda = Logger::getInstance();
3.      loggerSingletonEEzACsoda.log(ERROR, "Ha singleton ez ott kell legyen a fileban");
4.
5.      // If '=' operator and copy constructor wouldn't be disabled, this would
6.      // create another instance of Logger. Since these operations are deleted,
7.      // the compiler throws errors.
8.
9.      // Logger loggerSingletonEEzACsodaPart2 = Logger::getInstance();
10.     // Logger loggerSingletonEEzACsodaPart3 = logger;
```