**Raktár Manager program**

*Készítette: Heim József és Majoros Norbert*

Elvártak: Raktár Manager program tervezése, amely a következő funkcionalitások és alrendszerek megtervezett interfészeit és csontvázát kell tartalmaznia. Több inhomogén raktárrendszerrel való kommunikáció és közös menedzsment felület biztosítása. Áru, rendelés, beszállítók és vásárlók menedzsmentje. Ennek tartalmaznia kell ésszerű mennyiségű tervezési mintát.

Az elképzelésünkben úgy néz ki a feladat, hogy létezik kettő darab raktár (lehetett volna több is, mint ahogy a feladatban elhangzik) és ezek különböznek egymástól, oly módon, hogy az egyikben élelmiszereket tudunk eltárolni 50 db mennyiségig, amíg a másikban 150 db műszaki cikk tárolására van lehetőség maximálisan. A továbbiakban szó esik róla, hogy a raktárak készlete meg lesz jelenítve a képernyőn. Ehhez különböző színeket használunk: A készleten lévő termékek sárgával, a sikertelen műveletek(teli raktárba való beszúrási kísérlet, kért termék éppen nincsen raktáron) pirossal, a sikeres vásárlásokat zölddel, és az éppen eltárolt termékeket változatlanul szürkével fogjuk jelölni.

A műszaki cikkeknél megvalósult az az ötlet, hogy lehessen kiválasztani, hogy az adott termék vezeték- illetve vezeték nélküli legyen. Emellett a műszaki cikkek tárolására alkalmas raktárban hozzárendelünk a termékekhez egy 12 karakterből álló cikk számot is. Ebből kifolyólag a műszaki cikkek tárolására egy Dictionary-t használunk, ezzel szemben az élelmiszerek esetében nincs ilyen igény tehát azokat egy Listában tároljuk. Látható, hogy ez a két adatszerkezet különbözik egymástól feldolgozásuk szempontjából, emiatt itt bevezettük az **Iterátor** tervezési mintát. Erre azért volt szükség, hogy a Manager osztály átláthatóan, egyszerűen és ésszerűen legyen képes kezelni a raktárak készletét. Az Iterátor tervezési minta lényege, hogy segítségével szekvenciálisan érhetjük el egy objektum elemeit, a mögöttes megvalósítás megismerése nélkül. A mi esetünkben az igényt az idézte elő, hogy szeretnénk, ha a programunk minden egyes beérkezett vagy kiküldött áru esetén frissítve megjelenítené az aktuális készletet az össze raktárra nézve. Ezen funkció megvalósítására létrehoztunk egy *MonitorClass* osztályt (az elnevezés abból adódik, hogy a .NET keretrendszer alapértelmezetten rendelkezik Monitor osztállyal). Ez az osztály rendelkezni fog az adott raktár típusok egy-egy példányával amelyek megvalósításáról a későbbiekben lesz szó. Mindezek mellett ez az osztály tartalmaz két iterátor példányt is (egyet-egyet a raktárakhoz). Ezek segítségével képes végig haladni a raktárak készletein és megjeleníteni azokat a képernyőn.

A projekt feladatunkban elő került a **Factory** tervezési minta is. Ezzel a programtervezési mintával kiváltható a kódban elhelyezett számos ugyanolyan példányosító utasítás. Továbbá emelhetjük a termék absztrakciós szintjét, vagy elrejthetünk olyan információkat a kliens elől, amelyekre csak az objektum létrehozása miatt lenne szüksége. Kezdetben létrehoztuk a *SimpleProductFactory*-t ami egy Interface. Ezt implementálják a konkrét Factory osztályok: *ProvisionProductFactory*, *ElectricProductFactory*. Utóbbi osztályoknak az OrderProduct() metódusát fogják használni a különböző raktárak az áruk tárolásakor és elküldésekor. Ezek a raktárak implementációja fog most következni.

Van egy *Warehouse* nevezetű abstract osztályunk, ez olyan mezőket és metódusokat tartalmaz amelyek minden raktárnál megegyeznek például: Az áruk tárolása minden raktár esetén úgy zajlik, hogy az adatszerkezetébe belehelyezzük a terméket. Ebből az osztályból származnak le a konkrét áru típus tárolására alkalmas raktárak, amelyek az ősüket kiegészítik speciális metódusokkal amelyek csak az adott raktártípusra jellemző műveleteket írják le. Ezek a raktárak konkrétan a mi esetünkben: *ProvisionWarehouse*, *ElectricWarehouse*. A raktáraink tartalmaznak egy CreateIterator() metódust amire a *MonitorClass* osztályban lesz szükség a kiíráshoz. Ahogy már fentebb elhangzott a raktárak a nekik megfelelő Factory-nek az OrderProduct() metódusát fogják használni az áru kezelésekor, azonban mielőtt egy adott áru eltárolásra kerülne, végig fut egy vizsgálat ami meghatározza, hogy van-e hely neki a raktárban. Ugyan ez történik akkor is, ha venni szeretnénk egy terméket. Ilyenkor a vizsgálat tárgya az, hogy van-e raktáron a vásárolni kívánt áruból.

Termelők és Fogyasztók megvalósítása következőképpen történik. A fogyasztók reprezentálása egy egyszerű *Consumer* osztályban történik. Egy adott fogyasztó példánynak rendelkeznie kell minden olyan raktárnak egy példányával, amelyből vásárolni szeretne. A megvalósításához hozzátartozik többek között az is, hogy külön metódusokkal tudunk rendelni a különböző raktárakból. Ezzel szemben a termelők tevékenységét „szét kell választani”. Erre azért van szükség, mert egy gyártó általában egy terméktípust gyárt (például nem nagyon létezik Zanussi fehér kenyér). Az implementáció elején egy abstract *Producer* osztályt hozunk létre, ami csak az alapvető dolgokat valósítja meg (Megjegyzés: ilyen például az, hogy áru típusától függetlenül az árut becsomagolva szállítják. Tehát ez minden áru esetén azonos művelet). Ebből az osztályból származnak le a konkrét termék típusokat leíró osztályok. Ezek már tartalmaznak olyan metódusokat, amelyek már a konkrét termék létrehozásáért lesznek felelősek.

A fenti bekezdésekben elhangzott osztályok műveleteinek és kapcsolatainak kezelésére létrehoztunk egy *WarehouseManager* osztályt. Belső szerkezete a következő lesz: Tartalmazni fog az adott raktárakból egy-egy példányt az áruk kezeléséhez. Egyúttal tartalmazni fog egy *MonitorClass* objektumot amely segítségével minden változtatás után frissülni fog a készlet kiírása. Mindezek mellett tartalmaz termelőket és fogyasztót is. Az eddig felsoroltak segítségével ebben az osztályban definiálva van négy könnyen átlátható metódus, a különböző típusú termékek rendeléséhez és eladásához. Ez azért lesz hatékony, mert így a főprogramnak elég egy ilyen osztálybeli példányt létrehozni, ahhoz hogy tudjuk kezelni a raktárakat. A felhasználó ezáltal nem fog belátni a háttérkód bonyolult rendszerébe, csak ennek az osztálynak az átláthatóbb metódusait kell alkalmaznia.

