**Programozási technológiák beadandó**

Lehóczky Csaba

A beadandó feladat koncepciója egy raktármenedzselő program volt, amelynek a feladatban leírt funkcionalitások és alrendszerek megtervezett interfészeit, és "csontvázát" kell tartalmaznia. Ezeket a feladatokat az órai példaprogramok alapján, az internet és a megfelelő szakirodalom segítségével készítettem el.

A programban megtalálható egy fő-raktár és ebből származik kettő al-, egyedi-raktár amik a fő raktárral kommunikálnak, a kettő alraktárban telefonokat és táblagépeket tárolunk. Amikor egy készülék bekerül a rendszerbe, azaz a fő-raktárba a program eldönti melyikbe való és oda helyezi. A raktárból kétféle entitás tud rendelni: egy bolt vagy egy személy. Amit a program rögtön megvizsgál mind a kettő vásárlónál, hogy van-e elég pénze végrehajtani a tranzakciót, ha nincs a megfelelő kivételt dobja. Ha egy bolt szeretne vásárolni, ott, mint a való életben is van egy minimum vásárlási limit mind a kettő készüléktípusnál, ha ezt nem haladja meg a tranzakció akkor az nem hajtható végre, kivételt dob, ez a megkötés magánszemélynél nincs. A projektben számlát is tudunk kiállítani, az „Invoice” osztály segítségével, ezeken szerepel:

* a vevő neve
* a vevő címe
* a készülékből vásárolt mennyiség
* illetve maga a készülék neve és tulajdonságai.

A programnak nincsen felhasználói felülete, mivel a csontváz osztály elégnek bizonyult a beadandó dolgozat megírásához, így a megírt tesztek bizonyítják a működését.

A projektem a következő osztályokat, kivételeket illetve interfészeket tartalmazza:

Osztály:

* MainWarehouse: Ez a fő raktár. Implementálja az IObserverInfo interfészt. Singleton tervezési mintát használtam.
* PhoneWarehouse: Az egyik egyediraktár, telefonokat tartalmaz. Implementálja az IObserver és az IDeviceOperations interfészeket.
* TabletWarehouse: A második egyediraktár, a táblagépeket tartalmazza. Ez az IObserver és az IDeviceOperations-t implementálja.
* Device: A termék adatait, tulajdonságait tárolja.
* Customer : A vásárlók adatait tárolja.
* PersonCustomerStrategy: A személyekre vonatkozó rendelési stratégiát tartalmazza. Implementálja az ICustomerStategy interfészt.
* StoreCustomerStrategy: A boltokra vonatkozó rendelési stratégiát tartalmazza. Implementálja az ICustomerStategy interfészt.
* Order: A megrendeléseket és az azokhoz való adatokat tartalmazza.
* Invoice: A számlához szükséges adatokat tartalmazza.
* DeviceSupplier: A beszállítók adatait tartalmazza.

Kivételek:

* NoPaymentException: Ez a kivétel arra az esetre van ha a tranzakciót nem sikerült lebonyolítani, mert nem volt elég pénzmennyisége az entitásnak.
* MinCostException: Kivétel arra az esetre, ha a bolt túl alacsony összegben szeretett volna tranzakciót lebonyolítani.

Interfészek:

* ICustomerStrategy: A stratégiatervezési mintához létrehozott szükséges interfész.
* IObserverInfo: Az observer tervezési mintához szükséges interfész.
* IObserver: Az observer tervezési mintához szükséges interfész.
* IDeviceOperations: A készülékekkel kapcsolatos műveletekhez szükséges interfész.

A projektem egy **enum**-ot is tartalmaz Device TypeEnum néven, ez a termékek fajtáit tartalmazza.

Mivel a programnak nincs kezelő felülete így tesztekkel lehet megbizonyosodni a működéséről, a tesztek a következőek:

* + InvoiceTest: A számla kiadást és elkészítést teszteli.
  + NotPaymentExceptionTests: Ez a teszt azt vizsgálja, hogy a személyek és boltok, tudnak, vagy nem tudnak vásárolni, pénzmennyiség alapján tesztel.
  + MinCostExceptionTests: A boltoknál limitáltan lehet vásárolni, ez a minimum végösszeget teszteli.
  + OrderTests: A rendelés leadást és rendelés törlést teszteli.
  + WarehouseTests: Az egy fő raktárt és a kettő alraktárat teszteli.

A kritérium szerint tervezési mintákat kellett használni a beadandófeladat teljesítéséhez, én három tervezési mintát használtam fel a raktár menedzserprojektemben:

* Singleton
* Observer
* Stratégia

A **singleton**, ahogy a nevéből is következtethető olyan programtervezési minta, ami egyetlen bizonyos objektumra korlátozza az osztály létrehozható részeinek a számát. Ezt a projektemben a fő raktárnál, azaz a MainWarehouse-nál használatam, mert ez singleton-ként lett létrehozva, mivel csak egys példányra volt ebből szükségünk.

A második tervezési mint amit használtam az **observer** volt, melyben egy objektum, azaz alany egy listát vezet a megfigyelőkről, (alárendeltjeiről) és automatikusan értesíti őket, ha történt bármilyen nemű állapotváltozás, többnyire valamely metódusuk meghívásának segítségével. A programonban a fő raktár vezet listát az al-, egyediraktárokról és ha készülék érkezik a fő raktárba, akkor ez értesíti a két készüléktípushoz készült raktárat és bele rakja a megfelelőbe.

A **stratégia** minta lehetővé teszi, hogy egy algoritmus viselkedését a futás során válasszuk meg. Különböző viselkedéseket adhatunk meg más hozzá hasonló objektumoknak. A projektemben a stratégia mintát az entitások elválasztására használtam azaz, hogy magánszemély vagy bolt vásárol-e. A két vásárlói entitáskör annyiban tér el egymástól, hogy az áruházaknak mindkét készülékre van egy minimum végösszeg, amit el kell érni különben nem, teljesülhet a tranzakció, és kivételt dob a program.

**UML-ábra a projektemhez:**

