Programozási technológiák beadandó feladat

Vakarcs Tamás

UUQSB2

1. Feladat

Raktár manager program tervezése amelynek a következő funkcionalitások és alrendszerek megtervezett intefészeit, és "csontvázát" kell tartalmaznia:

- Több inhomogén raktárrendszerrel való kommunikáció, és közös menedzsment felület biztosítása.
- Áru menedzsment
- Rendelés menedzsment
- Beszállítók és Vásárlók menedzsmentje.

A rendszerben használt megoldásokról, és azoknak magyarázatáról minimum 3 oldalas dizájn dokumentum is szükséges az Elégséges (2) jegy eléréséhez. A jegyek a program elkészítésekor alkalmazott design patternek mennyisége, ésszerűsége, a leadott kódbázis minésége, valamint a dizájn dokumentumban leírtak, és a hallgató szóbeli elbeszélgetés során nyújtott beszámolójának összeségéből áll össze. A leadott kód igényességének illeszkednie kell a tárgyban elhangzott követelményeknek!

2. Működési elv és megvalósítás

Egy vállalat raktár manager programját készítem el. A vállalatnak két féle raktára van. Az alapanyagok raktára (RawMaterialWarehouse) és a késztermékek raktára (FinishedMaterialWarehouse). A vállalat 5 fő termék kategóriát gyárt: autó

alkatrészeket, számítógépeket, telefonokat, játékokat és televíziókat. Az alapanyagok a beszállító (Supplier) felől érkeznek. Amikor egy vevő (Customer) lead egy rendelést, az egy rendelési listába kerül ami alapján a termelés (Producer), az alapanyagokból készterméket állít elő amik először a késztermék raktárba kerülnek, ahonnan majd a vevő át tudja venni.

A SupplierController osztály végzi a beszállítandó alapanyagok beérkezését az alapanyag raktárba a Supply() metódussal.

Az OrderController osztály készíti el a vevő számára a rendelést. A MakeOrder() metódust meghívva, egy OrderBuilder osztályt példányosít amely elkészíti a megfelelő megrendelést, majd vissza tér egy Order példánnyal.

Minden rendeléshez tartozik egy rendelési lista, amely tartalmazza az összes megrendelt készterméket. Ezt a vevő az interneten tudja elkészíteni amit az OrderListProvider osztály tölt le a HttpAdapter osztály segítségével.

Ha a rendelés elkészült, a ProductionController osztály a Start() metódusával elkészíti a termékeket nyersanyag felhasználásával, majd ezeket a késztermék raktárba továbbítja.

Végezetül az elkészült rendelést a CustomerController osztály konstruktorának paraméterül adva, a Buy() metódusával a vevő meg tudja venni.

3. Felhasznált tervezési minták

Singleton (Egyke)

A FinishedProductWarehouse és a RawMaterialWarehouse osztályban használtam fel. Azért van rá szükség mert a raktárakból csak egy példányt szeretnék hogy létezzen a rendszerben, ami globálisan is elérhető. A private konstruktorra azért van szükség, hogy külön ne lehessen példányosítani. Kell egy statikus mező, aminek instance a neve az egyetlen példányunk számára. A getInstance() metódus az ami visszadja számunkra ezt a példányt amennyiben már létezik, ellenkező esetben előtte létre is hozza.

Ezek közül az egyik implementációja:

```
private static FinishedProductWarehouse instance = null;

private FinishedProductWarehouse(){}

public static synchronized FinishedProductWarehouse getInstance(){
    if (instance == null){
        instance = new FinishedProductWarehouse();
    }
    return instance;
}
```

Factory (gyár)

A FinishedProductFactory osztályban használtam fel. Különböző megvalósítású, de azonos felületű objektumok létrehozására van szükség. Könnyen bővíthető, és a klienseknek minimális ismeretre van szükségük. A FinishedProduct az az osztály, ami a termékek közös jellemzőjét tartalmazza, a különböző termékek osztályai pedig ebből származnak. A FinishedProductFactory tartalmazza a Create(String category) metódust, ami a termék kategóriája alapján létrehozza a megfelelő osztályt, és azzal tér vissza.

```
public class FinishedProductFactoryImpl implements FinishedProductFactory {
    @Override
    public FinishedProduct Create(String category) {
        FinishedProduct finishedProduct = null;
        switch(category)
        {
            case "Toy":
                finishedProduct = new Toy();
            break;

            case "Television":
                finishedProduct = new Television();
            break;

            case "Phone":
                finishedProduct = new Phone();
            break;

            case "CarPart":
                finishedProduct = new CarPart();
            break;

            case "Computer":
                finishedProduct = new Computer();
            break;
        }
        return finishedProduct;
    }
}
```

Builder (Építő)

Az OrderBuilder osztályban használtam fel. Egy építési folyamattal ugyan azt az objektumot különböző szerkezeti elemekkel hozhatom létre. Order példányokat tud létrehozni, 3 mezője és 4 metódusa van. A setCustomer, setDeadline és a setOrderList a mezők értékét állítja be. Végül a build() ezek alapján elkészíti az Order példányt és vissza tér vele.

```
public class OrderBuilderImpl implements OrderBuilder{
    private OrderBuilderImpl orderBuilder;

    private Customer customer;
    private Date deadline;
    private List<OrderElement> orderList;

    @Override
    public OrderBuilderImpl setCustomer(Customer customer)
    {
        this.customer = customer;
        return this;
```

```
    }
    @Override
    public OrderBuilderImpl setDeadline(Date deadline)
    {
        this.deadline = deadline;
        return this;
    }
    @Override
    public OrderBuilderImpl setOrderList(List<OrderElement> orderList)
    {
        this.orderList = orderList;
        return this;
    }
    @Override
    public Order build()
    {
        return new Order(customer,deadline,orderList);
    }
}
```

Adapter

A HttpAdapter osztályban használtam fel. Az OrderListProvider ezen keresztül tudja letölteni az internetről az adott vevőhöz tartozó rendelési listát. Egy metódusa van ami a get(String p_url), ami egy URL-t paraméterül kapva elvégez egy http lekérdezést, és visszatér az eredményével.

```
public class HttpAdapterImpl implements HttpAdapter {
    @Override
    public InputStream get(String p_url) throws IOException {
        URL url = new URL(p_url);

        HttpURLConnection connection =
(HttpURLConnection)url.openConnection();

        connection.setRequestProperty("accept", "application/json");

        return connection.getInputStream();
    }
}
```