**Programozási technológiák gy. beadandó**

**Design dokumentum**

**Készítette: Didics Patrik László,  
Pusztai Dominik Tamás**

**A projekt:**

A beadandó feladat egy raktár manager program tervezése volt, amelynek a következő funkcionalitások és alrendszerek megtervezett interfaceit, és "csontvázát" kellett tartalmaznia. A feladatot az órán szemléltetett tervezési minták segítségével kellett megoldani.

Alapkoncepciónkat a méltán híres afrikai-amerikai mérnök, vállalkozó, több neves cég közül a SpaceX és a Tesla elnökvezér-igazgatója, Elon Musk ihlette. Raktármanager programunkban helyet kapnak előre elkészített és egyedi kiegészítőkkel felszerelhető elektromos autók, ezen kívül minden közlekedési jármű, amire szükségünk lehet egy esetleges Marsra szálláskor.

**SOLID elvek megléte**

**OCP(Open-closed principle „Nyilt/zárt elv”):** Az Open-Closed Principle (OCP), magyarul a nyitva-zárt elv, kimondja, hogy a program forráskódja legyen nyitott a bővítésre, de zárt a módosításra. Eredeti angol megfogalmazása: „Classes should be open for extension, but closed for modification”.

Tegyük fel, hogy az űrhajókat szeretném megkülönböztetni funkcionalitásuk szerint. Ehhez csak bővítenem kell a kódot, nem kell módosítanom rajta.

**SRP (Single Responsibility Principle „Egyetlen felelősség elve”):** Az Egy felelősség - egy osztály alapelve (angolul: Single Responsibility Principle – SRP) azt mondja ki, hogy minden osztálynak egyetlen felelősséget kell lefednie, de azt teljes egészében. Eredeti angol megfogalmazása: „A class should have only one reason to change”, azaz „Az osztálynak csak egy oka legyen a változásra”.

Az osztályok és metódusok csak egy adott feladatot hajtanak végre. A programon belül ilyenre példa a OrderFactoryImpl-ben belüli metódusok.

**LSP(Liskov Substitutional Principle „Liskov helyettesítési elv”):** A Liskov-féle behelyettesítési elv, rövid nevén LSP, kimondja, hogy a program viselkedése nem változhat meg attól, hogy az ős osztály egy példánya helyett a jövőben valamelyik gyermek osztályának.

A projektben használt tervezési minták:

* Abstract Factory
* Observer
* Decorator
* Singleton
* Builder
* Strategy

A program a következő osztályokat tartalmazza:

* Storage:
  + - Cart(Implementálja a Collector inferace-t. Eltárolja a kosárban lévő termékeket)
    - Storehouse (Implementálja a Collector interface-t. Maga a raktár, ahova az egyes termékeket tároljuk.)
  + Products:
    - CarProductFactory, SpaceShipProductFactory( Implementálják a ProductFactory interface-t. Az autók és az űrjárművek létrehozásáért felelősek.)
    - AbstractFactory( Megvizsgálja a ProductFactory-be, hogy az egyes factory-k csontváza megfelelő-e.)
    - Tesla (Product interface-t implementálja. Egy Tesla alapdarab áráért és leírásáért felelős.)
    - Moonwalker, SpaceShip, ReusableSpaceShip(Product interface-t implementálják. Az egyes űrjárművek egyesével állítják be árukat és leírásukat.)
  + CustomeOrder:
    - Order
    - OrderFactoryImpl
    - Preparation
  + Management:
    - Manager
    - Observer
  + SpecialProducts:
    - SpecialTesla(Választható extrákkal kibővített autó extráit tűrolja, getter és setter metódusaikkal.)
    - SpecialTeslaBuilder(Az extrákhoz megadott értékeket állít be.)
    - SpecialTeslaEngineer(Összeállítja az extrás Tesla autót)
  + TeslaAndSpaceX:
    - SpaceXDecorator, TeslaDecorator ( Implementálják a Product interface-t. Decorator Design Pattern-ért felelősek.)
    - EngineSpaceX és -Tesla, PaintjobSpaceX és -Tesla, SelfDrivingSpaceX és -Tesla( A Tesla- és a SpaceXDecorator osztályokat
    - PlainProduct(implementálja a Product interface-t. A termékek karosszériáját tartalmazza, amire kerülnek a különféle kiegészítések.)

A program a következő interface-ket tartalmazza:

* + Collector(A Cart és a Storehouse osztályok implementálják. )
  + OrderFactory
  + Product( Az összes termék árának és leírásának beállításáért felelős.)
  + ProductFactory
  + SpecialTeslaPlan (A SpecialTesla osztály implementálja. Beállítja a hozzáadható extrákat az autóhoz.)
  + BuildSpecialTesla (Egy speciális Tesla autó egyes komponenseinek felépítését végzi.)

A program a következő teszteket tartalmazza:

* StorageTest(Több teszt egy osztályban, az egész raktárat leteszteli)
* SpecialTeslaTest(Builder Design Pattern tesztelése)
* SpaceXDecoratorTest és TeslaDecoratorTest(Decorator Design Pattern tesztelése)

**Tervezési minták kifejtése:**

**Factory:**

Létrehozási tervezési minta. Ezzel a mintával lehet szépen kiváltani a programunkban lévő rengeteg hasonló new utasítást. Egy metódus visszaadja a lehetséges osztályok egyikét, amelyeknek közös szuperosztálya van. Akkor használjuk, amikor nem tudjuk előre, hogy melyik osztály objektumra van szükségünk. Amikor az összes osztály azonos alosztály-hierarchiában van. Szinte minden objektum Factory osztállyal hozható lére. Például: ProductFactory

**Builder:**

A minta egy csomó más projektből készült objektumok létrehozására szolgál, akkor használjuk amikor létre akarunk hozni egy object-et amit egy másik object alkotott. Speciális Tesla autó builder a legalkalmasabb erre a példára. A különféle kocsi kiegészítők által, egy konkrét speciális autót tudunk létrehozni, amihez egyedi felnit rendelhetünk, kibővíthetjük motorjai számát, több ülést szereltethetünk be vagy akár a belső tér színét is személyre szabhatjuk.

**Singleton:**

Olyan osztályok, amelyeket csak egyszer van szükségünk példányosítani mert csak a metódusokra van belőlük szükség pl.: különböző factoryk.

**Abstract Factory:**

Az Absztrakt gyár (angolul: Absztrakt Factory) egy létrehozási tervezési minta, amely olyan objektumok gyártására jó, amelyek képesek egymással együttműködni, ennek megfelelően több létrehozásra alkalmas metódust tartalmaz.

**Decorator:**

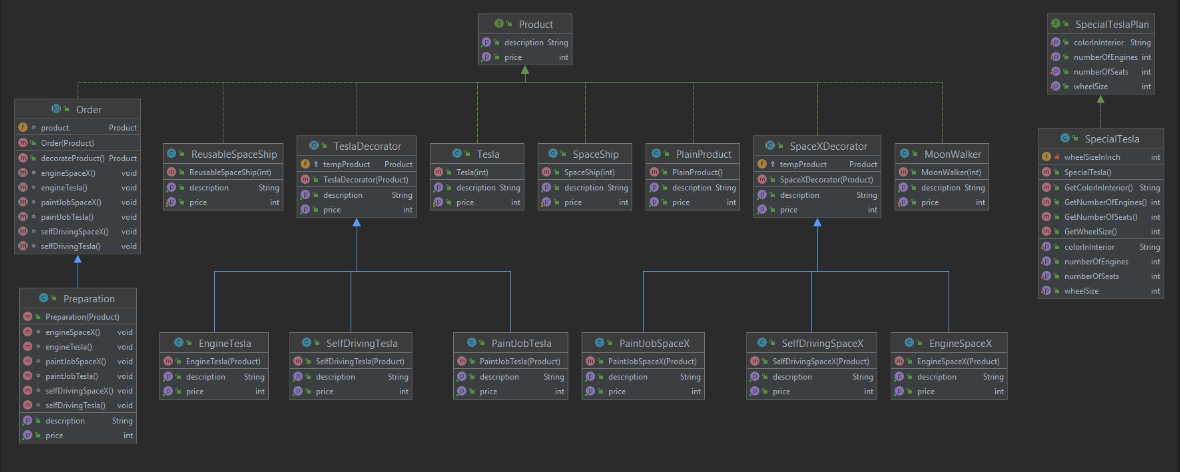
Lehetővé teszi adott objektumokhoz más viselkedések hozzáadását anélkül, hogy hatással lenne az azonos osztályból származó többi objektumra. A mi esetünkben a termék vázát képezte a karosszéria, amihez motort, fényezést és önvezető funkciót lehet hozzáadni, úgymondva „kidekorálni” vele az alap modellt. A termék váza a PlainProduct osztályba van eltárolva, ahol csak a leírása és az ára van eltárolva, ehhez jönnek hozzá a különböző kiegészítők.

**Observer:** A megfigyelő egy viselkedési tervezési minta, amely egy esemény által kiváltott változékony metódust emel ki egy egy-sok kapcsolat sok oldalára, és amely műsorszórással hívja meg a kiemelt metódusokat. A műsorszórás egy olyan programozási módszer, amikor egy metódus sok más, előre nem ismert, metódust hív meg egy lista alapján. Két fajtája van a húzó illetve a toló.

Részei:

* Alany: Tárolja a beregisztrált megfigyelőket, interfészt kínál a megfigyelők be- és kiregisztrálására valamint értesítésére.
* Megfigyelő: Interfészt definiál azon objektumok számára, amelyek értesülni szeretnének az alanyban bekövetkezett változásról. Erre a frissít (Update) metódus szolgál.

Esetünkben az OrderFactory implementálja az Observert, majd ez által hívhatók meg a metódusok.

*UML ábra:*

