# BPI1116L - OOP tervezési minták

## OOP

Az objektumorientált programozás egy programozási módszertan, mellyel a megvalósítandó feladatokat végző programok objektumok együttműködését szervezik, úgy, hogy azok mindegyike valamely osztály egyede. Az osztályok olyan hierarchiának részei, mely öröklődési kapcsolatokkal épül fel. Az objektumorientált programozás alapjai helyett, foglalkozzunk most az azokra épülő tervezési mintákkal.

A programozási feladatok megoldási folyamata gyorsabb, az előállított program biztonságosabb, ha a megoldást korábbi, hasonló feladatok megoldásainál bevált minták alapján állítjuk elő.

## Tervezési minták

A tervezési minták tipikus megoldások a szoftvertervezésben gyakran előforduló problémákra. Ezek olyan előre elkészített alaprajzok, amelyeket testre szabhatunk, hogy megoldjuk a kódban ismétlődő tervezési feladatokat, problémákat. Nem találhatunk meg egy tervezési mintát, és másolhatjuk be a programunkba, ahogy azt a más funkciókkal vagy könyvtárakkal tehetjük. A tervezési minta nem egy konkrét kódrészlet, hanem egy általános fogalom egy adott probléma megoldására. Követhetjük a tervezési minta részleteit, és implementálhatunk egy megoldást, amely illeszkedik a saját programunkhoz.

A tervezési mintákat gyakran összekeverik az algoritmusokkal, mert mindkét fogalom tipikus megoldásokat ír le néhány ismert problémára. Míg egy algoritmus mindig meghatároz egy világos lépéshalmazt, amellyel elérhető egy cél, addig egy tervezési minta egy magasabb szintű leírása a megoldásnak. Ugyanazon tervezési minta kódja, amelyet két különböző programra alkalmaznak, eltérő lehet.

Egy algoritmus analógiája a főzési recept: mindkettőnek világos lépései vannak a cél eléréséhez. Másrészről, egy tervezési minta inkább egy alaprajzhoz hasonlít: láthatjuk, mi lesz az eredmény és annak jellemzői, de a megvalósítás pontos sorrendje rajtunk áll.

Miből állnak a tervezési minták?

A legtöbb mintát nagyon formálisan írják le, hogy az emberek sok különböző kontextusban reprodukálhassák őket. Az alábbi részek általában jelen vannak majdnem minden tervezési minta leírásában:

* A minta célja, röviden leírja mind a problémát, mind a megoldást.
* A motiváció tovább magyarázza a problémát és a tervezési minta által lehetővé tett megoldást.
* Az osztályok szerkezete megmutatja a tervezési minta minden részét és azt, hogy hogyan kapcsolódnak egymáshoz.
* A kódpélda az egyik népszerű programozási nyelven könnyebbé teszi a tervezési minta mögötti ötlet megértését.
* Néhány tervezési minta katalógus más hasznos részleteket is felsorol, mint például a tervezési minta alkalmazhatósága, a megvalósítási lépések és a kapcsolatok más tervezési mintákkal.

## OOP tervezési minták összefoglalása

A tervezési minták az objektumelvű modellezést támogató minták, amelyeket az osztály diagram tervezése során alkalmazunk azért, hogy a modell újra felhasználható, könnyen módosítható, biztonságosan működő, és hatékony legyen, valamint nem utolsó sorban megfeleljen a **SOLID** elveknek.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Osztályozás | | CÉL | | |
| Hatókör | Osztály | **Létrehozási**  (gyártási, létrehozó) | **Szerkezeti**  (strukturális) | **Viselkedési**  (kapcsolatteremtő, működés) |
| Objektum | **Factory Method** (Gyártófüggvény) | **Adatpter**  ((Osztály) Illesztő) | **Interpreter**  (Értelmező)  **Tempalte Method** (Sablonfüggvény) |
| **Abstract Factory**  (Elvont gyár)  **Builder**  (Építő)  **Prototype**  (Prototípus)  **Singleton**  (Egyke) | **Adapter**  ((Objektum) Illesztő)  **Bridge**  (Híd)  **Composite**  (Összetétel)  **Decorator**  (Díszítő)  **Facade**  (Homlokzat)  **Flyweight**  (Pehelysúlyú)  **Proxy**  (Helyettes) | **Chain of Responsibility**  (Felelősséglánc)  **Command**  (Parancs)  **Iterator**  (Bejáró)  **Mediator**  (Közvetítő)  **Memento**  (Emlékeztető)  **Observer**  (Megfigyelő)  **State**  (Állapot)  **Strategy**  (Stratégia)  **Visitor**  (Látogató) |

## Létrehozási minták (Creational Patterns)

A létrehozási minták segítségével úgy hozhatunk létre objektumokat, hogy rendszerünk rugalmasabb, könnyen bővíthető, a meglevő osztályok könnyebben újra felhasználhatóak legyenek. Meg kell különböztetnünk gyártó és termék oldalt.

## 

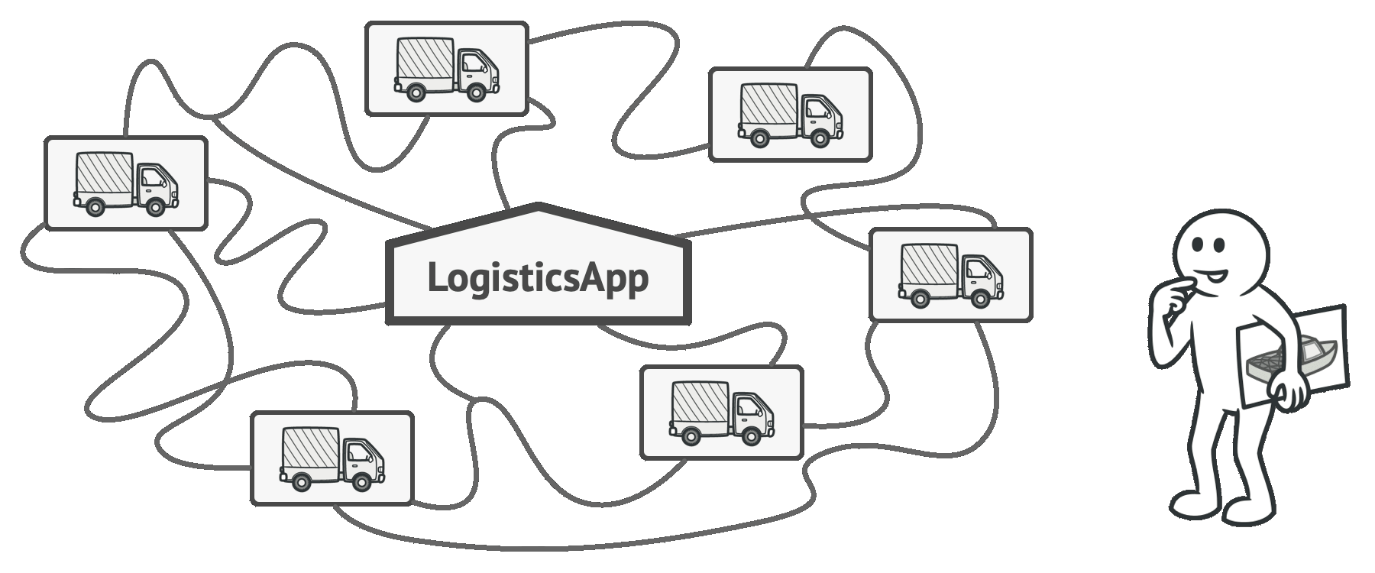
# Factory Method (Gyártófüggvény)

Ismert még: Virtuális Konstruktor

A **Factory Method** egy gyártó tervezési minta, amely felületet biztosít objektumok létrehozásához egy szuperosztályban, de lehetővé teszi az alosztályok számára, hogy megváltoztassák a létrehozandó objektumok típusát.

## Probléma

Képzeld el, hogy egy logisztikai menedzsment alkalmazást készítesz. Az alkalmazásod első verziója csak teherautók szállítását tudja kezelni, így a kódod nagy része a Teherautó osztályban található.



Új osztály hozzáadása a programhoz nem olyan egyszerű, ha a kód már összeköttetésbe került a meglévő osztályokkal.

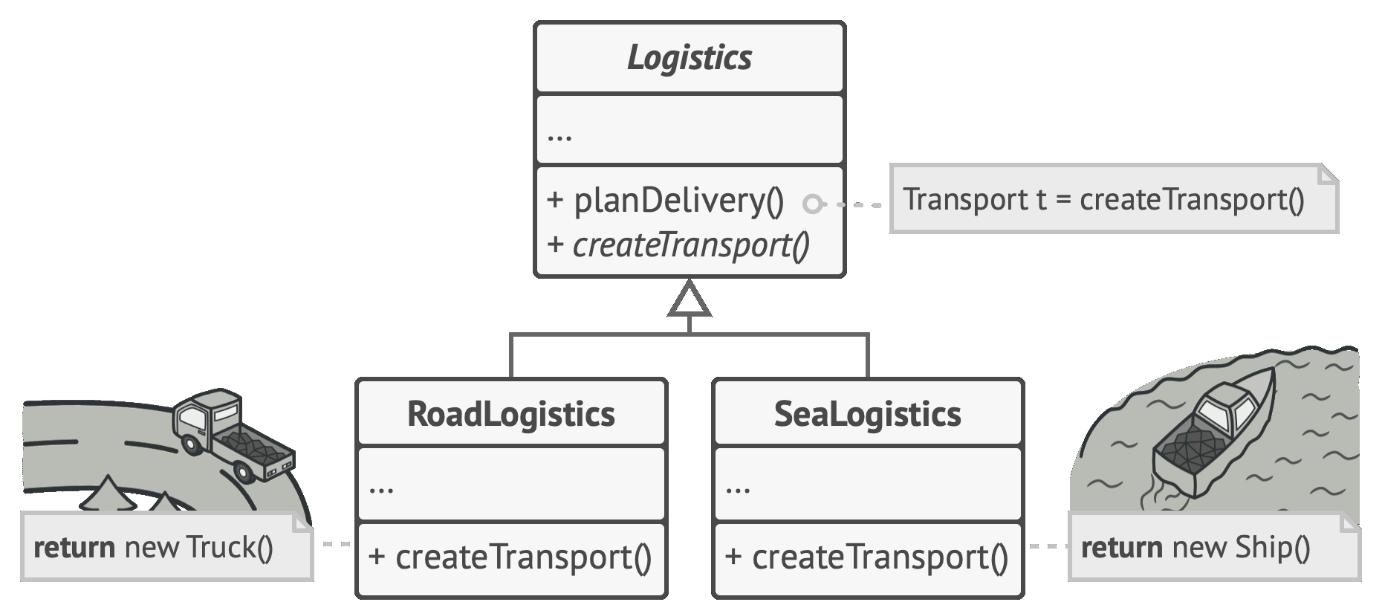
Egy idő után az alkalmazásod nagyon népszerűvé válik. Naponta tucatnyi kérelmet kapsz tengeri szállítási cégektől, hogy építsd be a tengeri logisztikát az alkalmazásba.

Ez jó hír igaz? Azonban felmerül a kérdés, a programkóddal kapcsolatosan: hogyan kerül átírásra, hiszen jelenleg a kódbázis nagy része összeköttetésben van a Teherautó osztállyal. A Hajó osztály hozzáadása miatt teljes átalakítást kell végezni a jelenlegi kódbázison.

Ráadásul, ha később úgy döntesz, hogy egy másik szállítási típust is hozzáadsz az alkalmazáshoz, valószínűleg mindezeket a változtatásokat újra el kell végezned. Ennek eredményeként elég csúnya kóddal fogsz rendelkezni a végén, amely tele van feltételes utasításokkal, amelyek az alkalmazás viselkedését váltják a szállítási objektumok osztályától függően.

## Megoldás

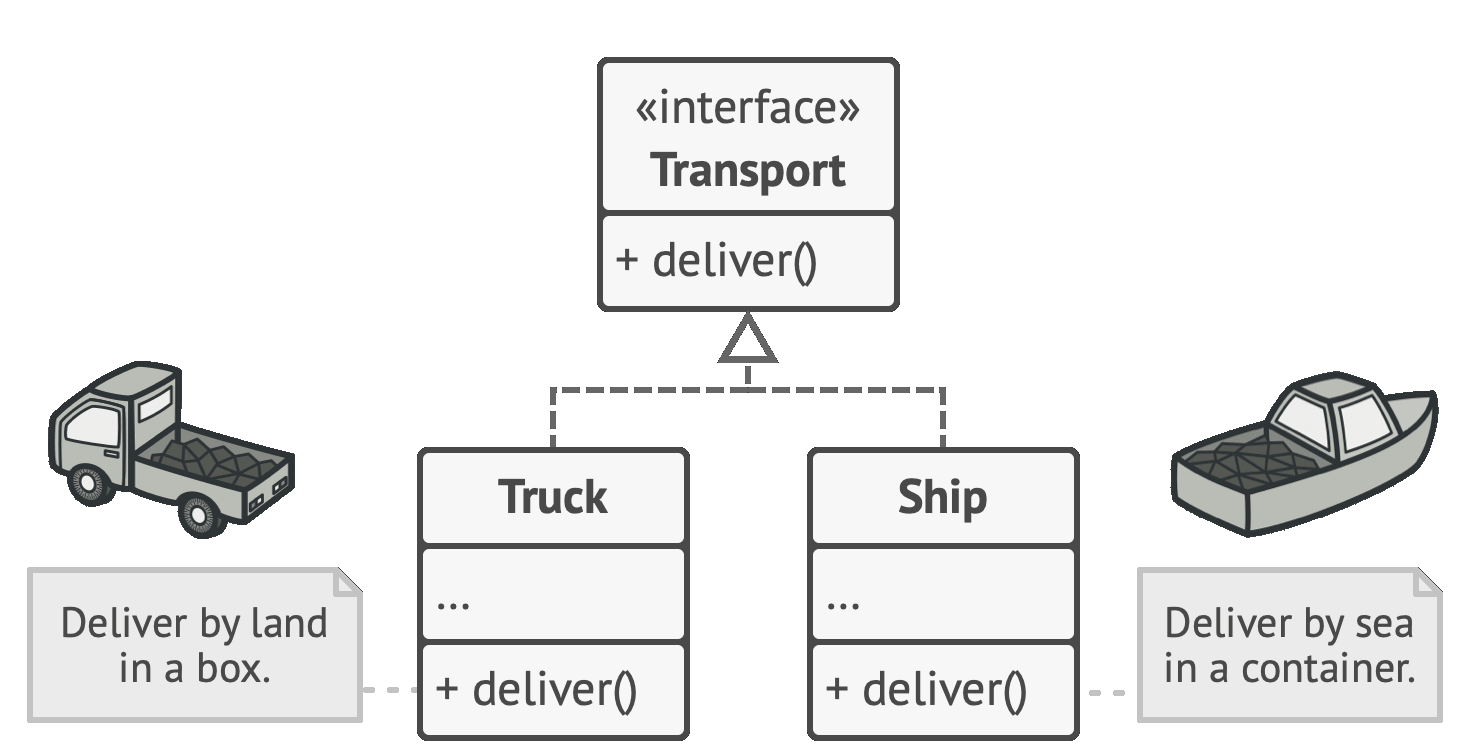
A **Factory Method** minta azt javasolja, hogy cseréld le a közvetlen objektumépítési hívásokat (az új operátor használatával) egy speciális gyártó metódus hívására. Ne aggódj: az objektumok még mindig az új operátor segítségével jönnek létre, de ez a gyártó metóduson belül történik. A gyártó metódus által visszaadott objektumokat gyakran termékeknek nevezik.



Az altípusok megváltoztathatják az objektumok osztályát, amelyeket a gyártó metódus ad vissza.

Első pillantásra ez a változás értelmetlennek tűnhet: csak áthelyeztük a konstruktor hívását a program egyik részéről a másikra. Azonban vedd figyelembe, hogy most felülírhatod a gyártó metódust egy altípusban, és megváltoztathatod a metódus által létrehozott termékek osztályát.

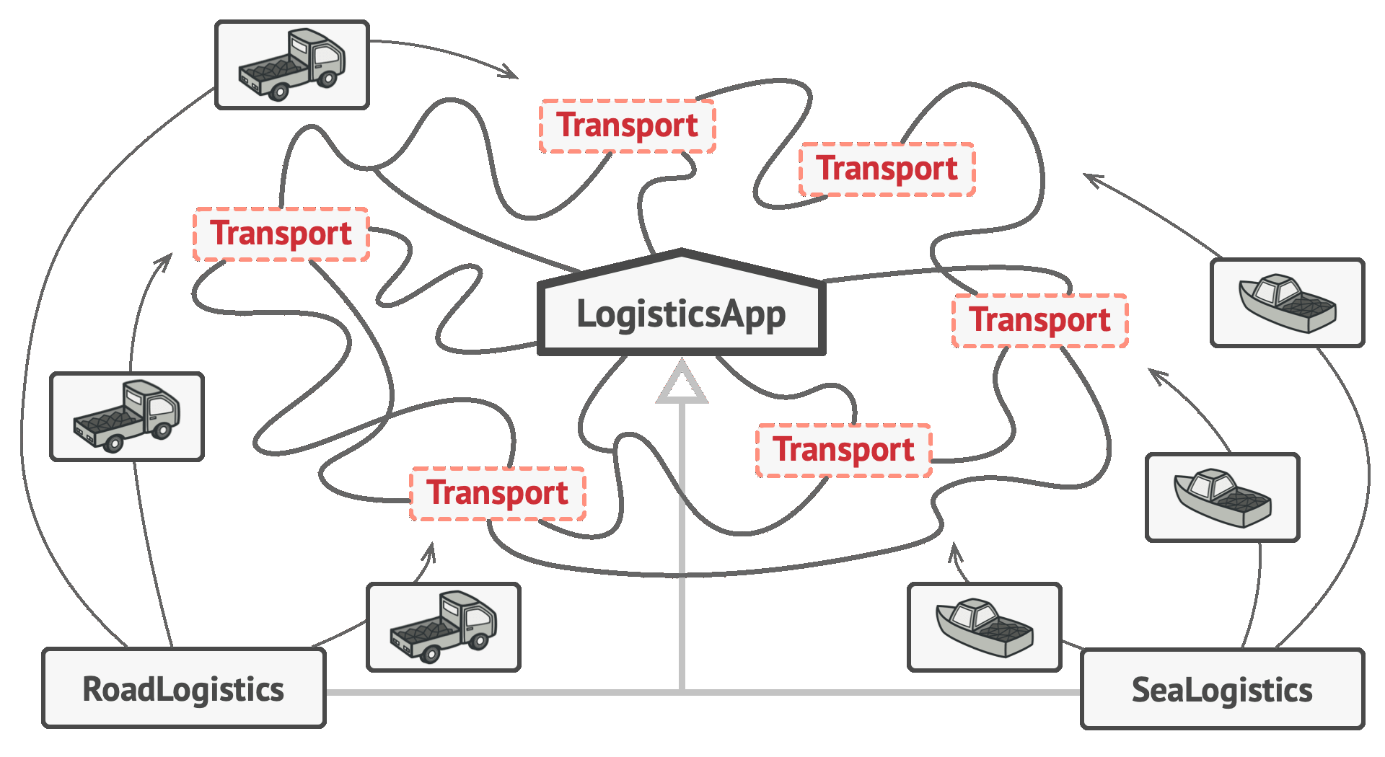
Van azonban egy kis korlátozás: az altípusok csak akkor adhatnak vissza különböző típusú termékeket, ha ezeknek a termékeknek közös alaposztálya vagy interfésze van. Ezenkívül az alaposztály gyártó metódusának visszatérési típusát ezen interfésznek kell deklarálnia.



Minden terméknek ugyanazt az interfészt kell követnie.

Például mind a Truck, mind a Ship osztálynak meg kell valósítania a Transport interfészt, amely egy deliver nevű metódust deklarál. Minden osztály másképp implementálja ezt a metódust: a kamionok szárazföldön szállítják a rakományt, a hajók tengeren. A RoadLogistic osztályban lévő gyártói metódus kamion objektumokat ad vissza, míg a SeaLogistic osztályban lévő gyártói metódus hajókat ad vissza.

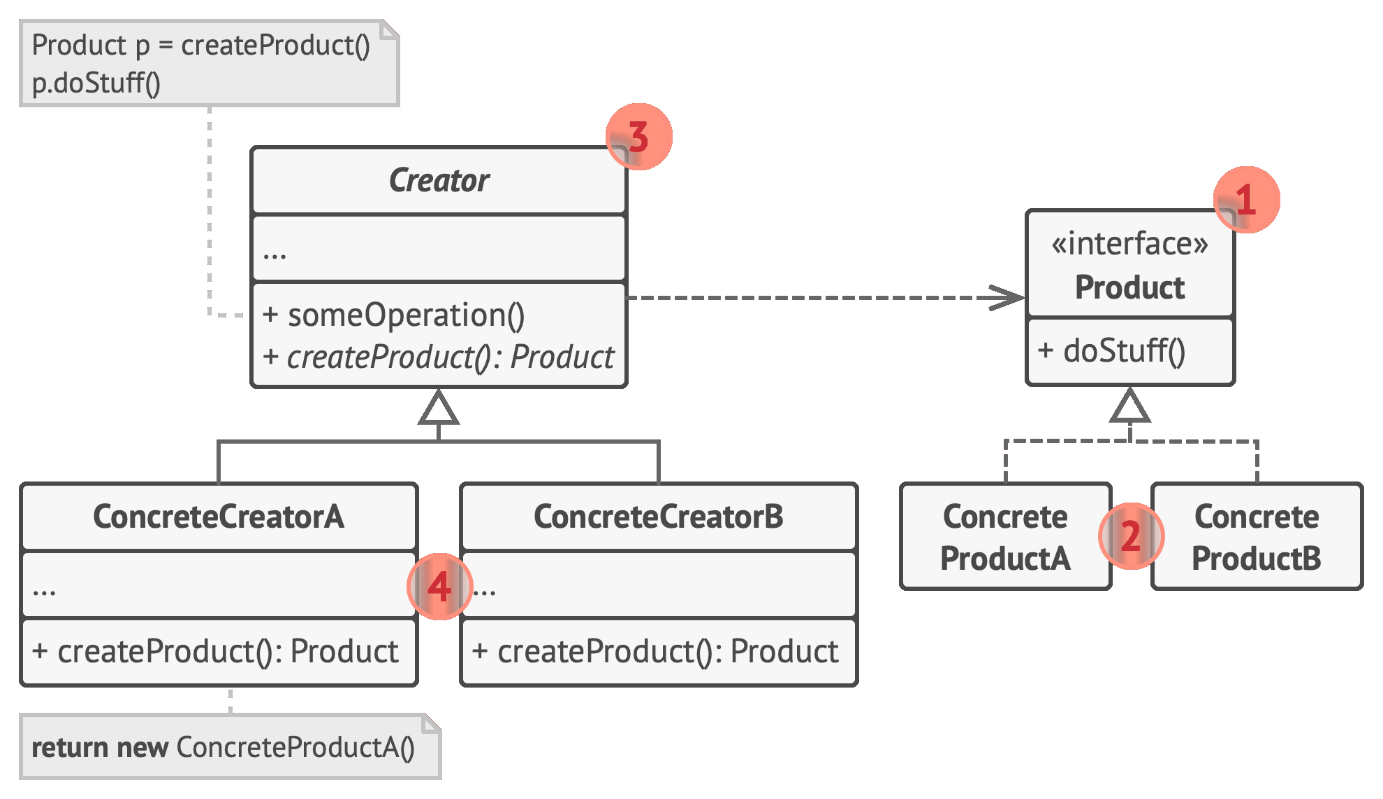
A gyártó metódust használó kód (gyakran *kliens*kód néven hívják) nem lát különbséget az egyes al-osztályok által visszaadott tényleges termékek között. A kliens az összes terméket absztrakt Transport osztályként kezeli.



Amíg minden termék osztály implementál egy közös interfészt, addig azok objektumait a klienskódhoz adhatod anélkül, hogy elrontanád a kódot.

A kliens tudja, hogy minden szállítási objektumnak kell rendelkeznie a deliver metódussal, de a pontos működése nem fontos a kliens számára.

### Struktúra



1. A **Termék** deklarálja az interfészt, ami közös minden olyan objektumhoz, amit a létrehozó és annak al-osztályai előállíthatnak.
2. A **Konkrét Termékek** különböző implementációi a termék interfésznek.

A **Létrehozó** osztály deklarálja a gyártói metódust, ami új termék objektumokat ad vissza. Fontos, hogy ennek a metódusnak a visszatérési típusa megegyezzen a termék interfésszel.   
  
A gyártói metódust absztraktként (abstract) is deklarálhatod, hogy kényszerítsd az összes alosztályt a metódus saját verzióinak implementálására. Alternatívaként az alap gyártói metódus visszaadhat valamilyen alapértelmezett termék típust.   
  
Megjegyzés, neve ellenére a termék létrehozása **nem** a létrehozó elsődleges felelőssége. Általában a létrehozó osztálynak már van némi alapvető üzleti logikája, ami a termékekhez kapcsolódik. A gyártói metódus segít ennek a logikának a leválasztásában a konkrét termék osztályoktól. Itt van egy hétköznapi példa: egy nagy szoftverfejlesztő cégnek lehet egy képzési osztálya a programozóknak. Azonban a cég egészének elsődleges funkciója még mindig a kódírás, nem a programozók előállítása.

1. A **Konkrét Létrehozók** felülírják az alap gyártói metódust, így az egy másik típusú terméket ad vissza.

## 

# Abstract Factory (elvont gyár)

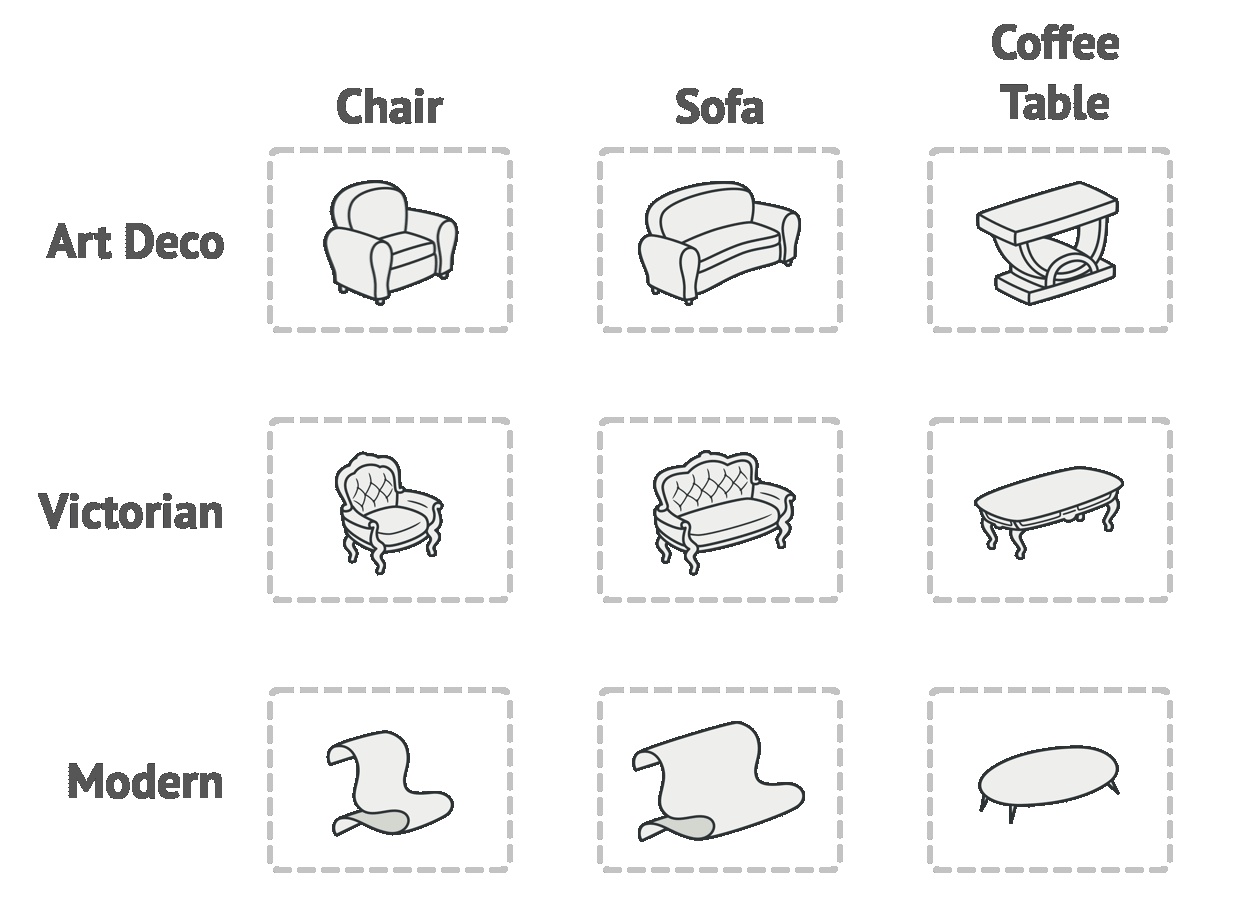
Az **Abstract Factory** egy gyártó tervezési minta, amely lehetővé teszi, hogy kapcsolódó objektumok családjait hozzuk létre anélkül, hogy meghatároznánk a konkrét osztályaikat.

## Probléma

Képzeld el, hogy egy bútorbolt szimulátort hozol létre. A kódod az alábbi osztályokat tartalmazza:

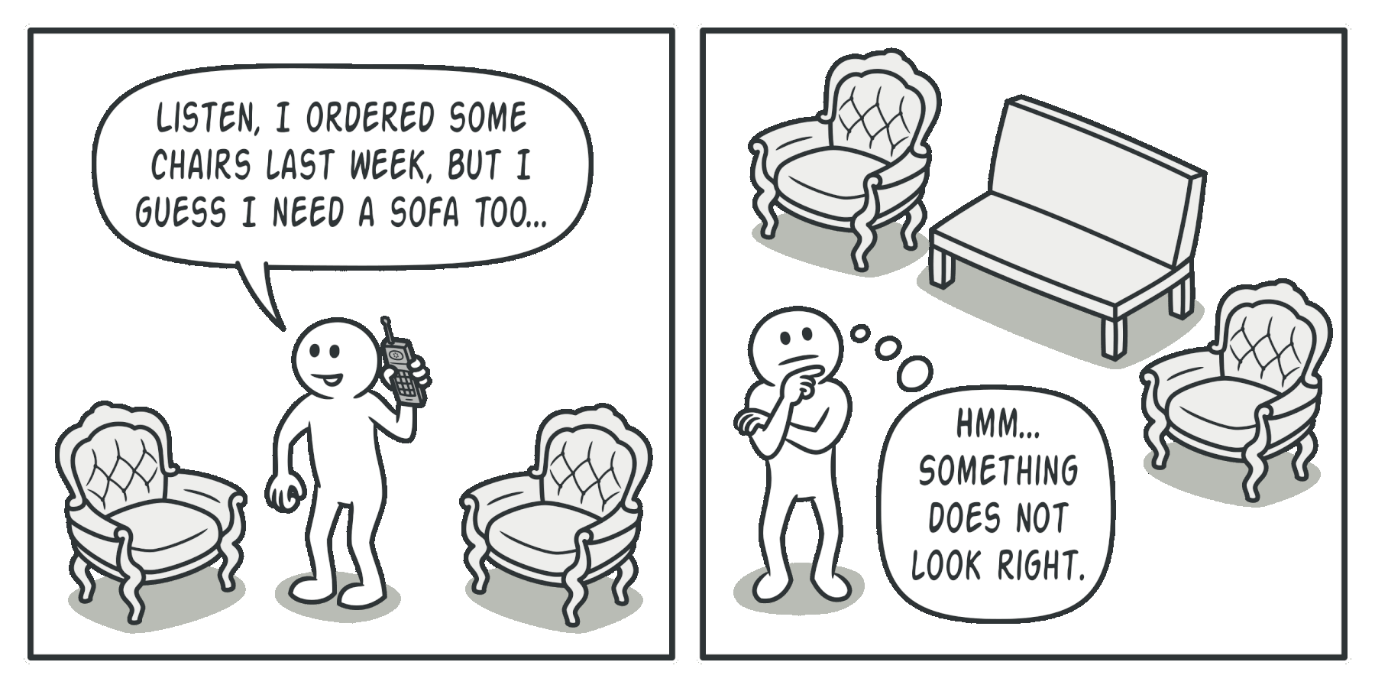
Egy kapcsolódó termékek családját, például: Szék + Kanapé + Dohányzóasztal.

Ennek a családnak több változata. Például a Szék + Kanapé + Dohányzóasztal termékek a következő változatokban érhetők el: Modern, Viktórián, ArtDeco.



Termékcsaládok és változataik

Szükséged van egy módszerre az egyes bútordarabok létrehozásához úgy, hogy azok összhangban legyenek a család többi tagjával. Az ügyfelek nagyon dühösek lesznek, ha nem összeillő bútorokat kapnak.

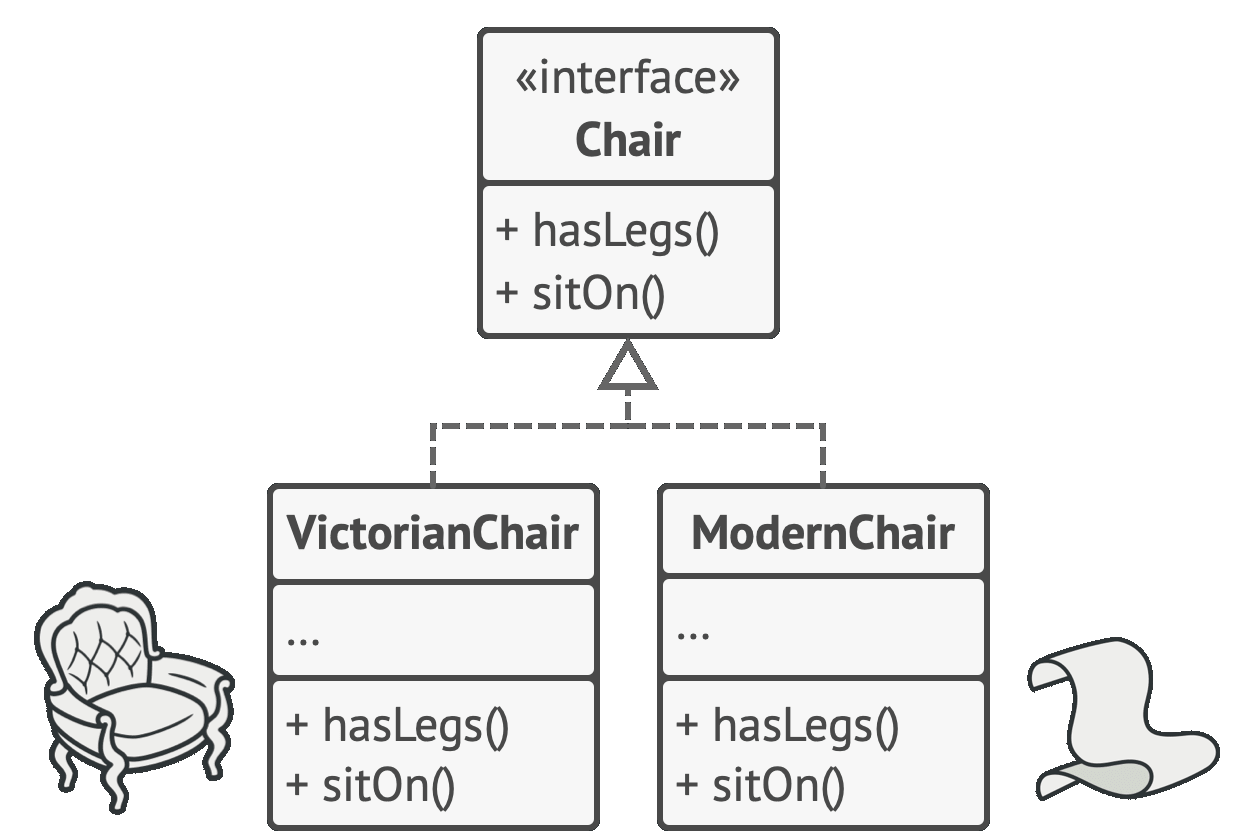


A modern stílusú kanapé nem illik a Viktórián stílusú székhez

Ezenkívül nem szeretnéd megváltoztatni a meglévő kódot, amikor új termékeket vagy termékcsaládokat adsz hozzá a programhoz. A bútorüzletek nagyon gyakran frissítik katalógusaikat, és nem szeretnéd, ha minden alkalommal meg kellene változtatnod az alapkódot.

## Megoldás

Az **Abstract Factory** minta javaslata, hogy deklarálj kifejezett interfészeket minden egyes termékhez a termékcsaládból (például szék, kanapé vagy dohányzóasztal). Ezután minden termékvariánst követhetsz ezekkel az interfészekkel. Például, minden szék variáns implementálhatja a Chari interface-t; minden dohányzóasztal variáns implementálhatja a CoffeTable interface-t, és így tovább.



Az objektum összes változatát egyetlen osztályhierarchiába kell tárolni.

A következő lépés az **Abstract Factory** deklarálása - egy interfész, amely tartalmazza a termékcsaládhoz tartozó összes termék létrehozásának módszereit (például createChair, createSofa és createCoffeeTable). Ezeknek a módszereknek vissza kell adniuk az **absztrakt** terméktípusokat, amelyeket a korábban kinyert interfészek képviselnek: Chair, Sofa, CoffeeTable és így tovább.

Mi a helyzet a termékvariánsokkal? Minden egyes variáns esetében egy termékcsaládnál, létrehozunk egy külön gyártó osztályt az AbstractFactory interfész alapján. A gyár egy olyan osztály, amely adott típusú termékeket ad vissza. Például a ModerFurnitureFactory csak ModernChair, ModernSofa és ModernCofeeTable objektumokat hozhat létre.