Tervezési Minták

Mi is az a tervezési minta?

A tervezési mintákat leginkább a **sportstratégiákhoz** vagy egy **kulináris alaprecepthez** hasonlíthatjuk.

Képzeld el, hogy egy futballezdő vagy. Ismersz egy bevált felállást (pl. 4-4-2), ami egy "minta" a védekezés és támadás megszervezésére. Ez azonban nem egy merev utasítássorozat (mint például: "fuss 5 métert, majd rúgd a labdát balra"), hanem egy keretrendszer. A pályán lévő játékosoknak a helyzethez kell igazítaniuk a stratégiát.

A programozásban is így működik: a minta nem egy kész kódrészlet, amit "copy-paste"-elhetsz, hanem egy bevált stratégia egy gyakori probléma megoldására, amit a saját programod nyelvére és igényeire kell szabnod.

Miben más, mint egy algoritmus?

- **Algoritmus:** Olyan, mint egy **GPS útvonalterv**. Pontosan megmondja, mikor kanyarodj jobbra vagy balra, hogy eljuss A-ból B-be.
- **Tervezési minta:** Olyan, mint egy **várostérkép a főbb csomópontokkal**. Tudod, hogy a folyón át kell kelned (ez a megoldandó probléma), és a térkép javasol néhány hídtípust (a minta), de neked kell eldöntened, melyiket építed meg és milyen anyagból.

1. Adapter (Adapter)

A koncepció

Ez a minta a "tolmács" szerepét tölti be két olyan rendszer között, amelyek egyébként nem értenék meg egymást. Lehetővé teszi az együttműködést inkompatibilis interfészek között anélkül, hogy a meglévő kódokat át kellene írni.

Új életszerű példa: A memóriakártya-olvasó

Képzeld el, hogy készítettél néhány fotót a profi fényképezőgépeddel, ami egy SD kártyára menti a képeket. A laptopodon azonban nincs SD kártya bemenet, csak USB-C portok vannak.

- Az SD kártya (a külső szolgáltatás) és a laptop (a te programod) nem kompatibilisek.

- A megoldás egy **kártyaolvasó** (az Adapter). Az egyik oldalán fogadja az SD kártyát, a másik oldalán USB-C csatlakozója van. A laptopnak csak az USB szabványt kell ismernie, a kártyaolvasó pedig elvégzi a "fordítást" a háttérben.

Szoftveres példa

Van egy webshopod, ami egy modern számlázó rendszert használ. Szeretnél bevezetni egy régi, de megbízható banki fizetési kaput (legacy system).

- A te rendszered `Customer` objektumokat használ (Név, Email).
- A régi banki rendszer viszont csak nyers `XML` adatokat fogad.
- **Megoldás:** Írsz egy Adapter osztályt, ami átveszi a `Customer` objektumot, kinyeri belőle az adatokat, XML-é alakítja, és továbbítja a banknak. A webshopodnak így nem kell XML-el bajlódnia, a banki rendszerhez pedig nem kell hozzányúlni.

2. Prototípus (Prototype)

A koncepció

Ez a minta arról szól, hogyan hozzunk létre új objektumokat egy meglévő példány lemásolásával (klónozásával), ahelyett, hogy nulláról építenénk fel őket. Ez különösen akkor hasznos, ha egy objektum létrehozása bonyolult vagy erőforrás-igényes.

Új életszerű példa: A dokumentum sablon

Gondolj egy céges szerződésre. Amikor egy új ügyfél érkezik, nem kezded el a szerződést üres lappal gépelni (fejléc, jogi nyilatkozatok, formázás). Ehelyett van egy **kitöltött minta-szerződésed** (a Prototípus). Ezt a dokumentumot lemásolod ("Mentés másként"), és csak a releváns részeket (ügyfél neve, dátum) írod át. A formázás, a bekezdések és a stílus már készen vannak, nem kell újra beállítanod őket.

Szoftveres példa

Egy stratégiai játékot készítesz, ahol hadseregeket lehet toborozni.

- Egy "Ork Harcos" létrehozása bonyolult: be kell tölteni a 3D modellt, a textúrákat, beállítani az életerőt, a sebzést és a felszerelést.
- Ha 100 orkot akarsz a pályára tenni, nem akarod mind a 100-nál végigcsinálni az inicializálást (betölteni a fájlokat a lemezről).
- **Megoldás:** Létrehozol *egyetlen* tökéletesen beállított Orkot. A többi 99-et ennek az egynek a `klónozásával` hozod létre a memóriában. Ez sokkal gyorsabb, és a klónok azonnal öröklik a beállításokat.

3. Felelősséglánc (Chain of Responsibility)

A koncepció

Ez a minta egy láncolatot hoz létre a kérések kezelésére. Ha egy kérés beérkezik, az végighalad a láncon. Minden egyes láncszem eldönti: "Tudok ezzel foglalkozni?". Ha igen, megoldja (vagy megoldja és továbbadja). Ha nem, továbbadja a következő láncszemnek.

Új életszerű példa: Céges költségelszámolás

Egy nagyvállalatnál dolgozol és vásárolni szeretnél valamit a cég számára. A jóváhagyási folyamat egy lánc:

1. **Csoportvezető:** Ha a tétel 50.000 Ft alatt van, ő azonnal jóváhagyhatja. Ha drágább, továbbküldi a kérést.
2. **Osztályvezető:** Ha a tétel 500.000 Ft alatt van, ő dönt. Ha ennél is drágább, továbbküldi.
3. **Vezérigazgató:** Ő dönt minden 500.000 Ft feletti tételről. Neked (a kérelmezőnek) nem kell tudnod, ki fogja végül aláírni. Te csak beadod a kérvényt a lánc elején (a csoportvezetőnek), és a rendszer eljuttatja az illetékeshez.

Szoftveres példa

Grafikus felhasználói felületek (GUI) eseménykezelése. Képzeld el, hogy van egy ablakod, abban egy panel, a panelen pedig egy "Mentés" gomb.

- Amikor rákattintasz a gombra, létrejön egy "Klikk" esemény.
- Először a **Gomb** kapja meg a lehetőséget. Ha a gomb le van tiltva (disabled), továbbpasszolja az eseményt.
- A következő a sorban a **Panel**. Lehet, hogy a panelnek van egy funkciója, ami minden kattintást naplóz.
- Végül az **Ablak** is megkaphatja az eseményt. Ez a lánc biztosítja, hogy az esemény ott kerüljön feldolgozásra, ahol a leginkább releváns, anélkül, hogy a gombnak tudnia kellene az ablak létezéséről.