Programtervezési minták

A programtervezési minták (design patterns) olyan általánosított megoldások, amelyek jól bevált módszereket kínálnak gyakori szoftvertervezési problémák megoldására. Ezeket a mintákat három fő csoportba sorolhatjuk, az alapján, hogy a szoftver amely aspektusát célozzák meg: **Kreációs**, **Szerkezeti** és **Viselkedési** minták.

1. Kreációs minták

Ezek a minták az objektumok létrehozásával kapcsolatos problémákra adnak megoldást, biztosítva, hogy az objektumok létrehozása rugalmas és hatékony legyen.

Főbb kreációs minták:

- **Singleton**: Biztosítja, hogy egy osztályból csak egy példány létezzen, és globális hozzáférést nyújt hozzá.
- **Factory Method**: Egy közös interfészen keresztül határozza meg, hogy melyik konkrét osztály példányosítása történjen.
- **Abstract Factory**: Több kapcsolódó vagy függő objektum családjának létrehozását teszi lehetővé anélkül, hogy azok konkrét osztályait meghatároznánk.
- **Builder**: Komplex objektumok lépésenkénti létrehozására szolgál.
- **Prototype**: Az objektumok másolásán alapuló létrehozási mechanizmust nyújt.

2. Szerkezeti minták

Ezek a minták az objektumok közötti kapcsolatokra és a program szerkezetére összpontosítanak, elősegítve az újrafelhasználhatóságot és rugalmasságot.

Főbb szerkezeti minták:

- **Adapter**: Egy meglévő osztály interfészét alakítja át, hogy kompatibilis legyen egy másik interfésszel.
- **Decorator**: Dinamikusan bővíti egy objektum funkcionalitását anélkül, hogy megváltoztatná annak osztályát.
- **Proxy**: Helyettesít egy másik objektumot, ellenőrizve vagy kiegészítve annak hozzáférését.
- **Composite**: Az objektumokat fastruktúrába szervezi, hogy az egyes és az összetett objektumokat egységesen kezelhessük.
- **Bridge**: Elválasztja egy objektum absztrakcióját a megvalósításától, lehetővé téve azok független változását.

3. Viselkedési minták

A viselkedési minták az osztályok és objektumok közötti kommunikációval, valamint együttműködésükkel foglalkoznak.

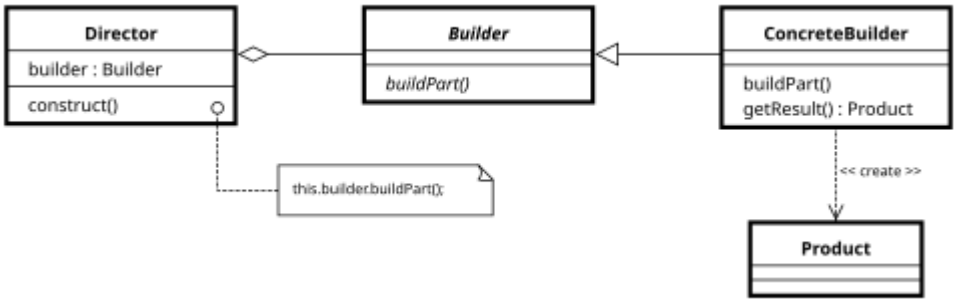
Főbb viselkedési minták:

- **Observer**: Egy objektum állapotának változásait értesíti más, tőle függő objektumoknak.
- **Strategy**: Egy algoritmus több változatának definiálására szolgál, amelyeket futásidőben cserélhetünk.
- **Command**: Műveleteket objektumokba csomagol, lehetővé téve a műveletek paraméterezését, mentését vagy visszavonását.
- **State**: Egy objektum viselkedése megváltozik az állapotának megfelelően.
- **Chain of Responsibility**: Egy kérést több objektum is kezelhet egymás után, amíg valamelyik meg nem oldja azt.

Pár fontosabb programtervezési minta

- Építő minta (Builder pattern):

Az Építő tervezési minta célja az összetett objektumok kialakításának és reprezentációjának szétválasztása. Ezáltal ugyanaz az építési folyamat különböző megvalósításokat eredményezhet.



Előnyei: Lehetővé teszi a termék belső reprezentációjának megváltoztatását.

- Magába zárja a konstrukciót és a reprezentációt.
- Lehetővé teszi az építési folyamat lépésenkénti ellenőrzését.
- Minden termékhez más Concrete Builder kell.
- Az építő osztályoknak megváltoztathatóknak kell lenniük.

- Lusta inicializáció (Lazy initialization)

A számítógépes programozásban, a lusta inicializáció programtervezési minta egy olyan taktika, amely szerint késleltetjük egy objektum létrehozását, vagy valamely számításigényes művelet elvégzését egészen addig, amíg az objektumra vagy a számítás eredményére először ténylegesen szükség lesz.

Példa JavaScriptben

```
var Fruit = (function() {
  var types = {};
  function Fruit() {};

  // visszaadja a paraméterként kapott objektum
  // tulajdonságainak(property) számát
  function count(obj) {
    return Object.keys(obj).length;
  }

  var _static = {
    getFruit: function(type) {
      if (typeof types[type] == 'undefined') {
        types[type] = new Fruit;
      }
      return types[type];
    },
    printCurrentTypes: function () {
      console.log('A létrehozott példányok száma: ' + count(types));
      for (var type in types) {
        console.log(type);
      }
    }
  };

  return _static;

})();

Fruit.getFruit('Apple');
Fruit.printCurrentTypes();
Fruit.getFruit('Banana');
Fruit.printCurrentTypes();
Fruit.getFruit('Apple');
Fruit.printCurrentTypes();
```

- Egyke (Singleton pattern)

  Az egyke programtervezési minta olyan programtervezési minta, amely **egyetlen objektumra korlátozza** egy osztály létrehozható példányainak számát.

Gyakori, hogy egy osztályt úgy kell megírni, hogy csak egy példány legyen belőle. Ehhez jól kell ismerni az objektumorientált programozás alapelveit. Az osztályból példányt a konstruktorával lehet készíteni. Ha van publikus konstruktor az osztályban, akkor akárhány példány készíthető belőle, tehát **publikus konstruktora nem lehet az egykének.** De ha nincs konstruktor, akkor nem hozható létre a példány, amin keresztül hívhatnánk a metódusait. **A megoldást az osztályszintű (statikus) metódusok jelentik.**

Egyszerű példa Java nyelven

```
public final class Egyke {

    private static volatile boolean letrehozva = false;
    private static volatile Egyke peldany = null;

    private Egyke() {}

    public static Egyke aPeldany() {
        if (!letrehozva) {
            synchronized(Egyke.class) {
                if (!letrehozva) {
                    peldany = new Egyke();
                    letrehozva = true;
                }
            }
        }
        return peldany;
    }
}
```