Storcz Tamás

Senior Fejlesztő
Crosssec Solutions Kft.

Az OOP

Objektumorientált programozás

Motiváció

Hatékony programozás

Közös nyelv és a közös gondolkodási séma (kivel?)

Olvasható kód

Újrahasznosítás biztonságosan

Programozással töltött időben az olvasás és írás aránya

10:1

Monolitikus – Moduláris példa (DB)

Modellezés - Osztályok

Az emberi gondolkodás mintájára osztályozás.

Cél

A valóságnál egyszerűbb, zárt környezet létrehozása, melyben a feladat *megfelelően* jól (idő, eszköz, pontosság) oldható meg.

Modell

A valóság egyszerűsített, idealizált értelmezése.

Modell tervezés:

A számunkra lényeges tulajdonságok, osztályozások, kapcsolatok meghatározása.
Algoritmusok, adatstruktúrák meghatározása.

Objektumorientált Programozás

Az **objektumorientált programozás** egy programozási módszertan. Ellentétben a korábbi programozási módszertanokkal, nem a *műveletek* megalkotása helyett az *egymással kapcsolatban álló* ***programegységek*** *hierarchiájának* megtervezése áll a középpontban.

Osztály Logikailag összetartozó adatok és műveleteik gyűjteménye.

Objektum Egy osztály egy példánya.

Interfész Egyirányú kapcsolódási felület strukturális definíciója. Meghatározza az interfészt közzétevő *programegység* használatának módjait.

crosssec SOLUTIONS

Alapelvek

Objektumorientált Programozás

OOP ALAPELVEK JELENTÉSE

Egységbezárás

A logikailag összefüggő adatokat és a rajtuk végzendő műveleteket egy egységbe zárjuk

Öröklődés

A származtatott osztályok öröklik az ősosztályok tulajdonságait és metódusait, de újabbakat is definiálhatnak

Polimorfizmus

Egy metódus azonosítója közös egy adott osztály hierarchián belül, ugyanakkor a hierarchia osztályaiban a metódus implementációja az adott osztályra nézve specifikus

OOP ALAPELVEK KIALAKÍTÁSÁNAK OKAI

Egységbezárás

Absztrakció, az adatrejtéssel/hozzáférési szintek szabályozásával a konzisztencia biztosítása

Öröklődés

Biztonságos újrahasznosítás, absztrakciós szint változtatás (szűkítés)

Polimorfizmus

Szűkülő absztrakciós szint specializálódó feladatai, újrahasznosítás kivételei

Egységbe zárás

OOP alapelvek

Egységbe zárás

Az összefüggő adatokat és az azokat kezelő műveleteket egy egységként kezeljük, csak a működtetéshez szükséges felületet biztosítjuk.

Előnyei

- Absztrakció
  Az implementációt interfész mögé rejtjük.
  Nem szükséges a belső működés ismerete, de nincs is rá igény
- Konzisztencia
  Ellentmondás-mentes (belső) állapotleírók, szabályos állapot átmenetek

Egységbe zárás - Konzisztencia

Adatrejtés: Mely adatokat rejtsük el? – `Private` vs. `Public`

Mindet, hogy a hozzáférés valóban absztrakt maradhasson.

Ezen az absztrakt interfészen keresztül **SZABÁLYOZOTTAN** használhatjuk az erőforrást.

Szabályozott hozzáférés példa (statikus DB)

Újrahasznosítás I - Osztályok

Az azonos típusú (tulajdonságú és viselkedésű) objektumok sablonjai, a kód újrahasznosítás első lépcsőfoka.

Az osztályok **példány**ai az objektumok.

- saját tulajdonság értékek → saját állapotok
- közös metódusok
- egységes, szabályozott hozzáférés → konzisztens működés

Osztály példa (DB)

Absztrakció – Struktúra vs. Objektum

Logikai egységet alkotó adatok és viselkedések (metódusok, események) gyűjteménye. Kevésbé modern nyelvekben csak adatok.

Konkrét változó példányok létrehozásának sablonja.

- A `struct` érték típusú – klasszikus értelemben változó
- A `class` referencia típusú – klasszikus értelemben objektum

Absztrakció példa (2D pont)

Öröklés

OOP alapelvek

Újrahasznosítás II – öröklés

A meglévő osztályokból származtatott újabb osztályok öröklik a definiálásukhoz használt ősosztályok tulajdonságait és metódusait, ugyanakkor újabbakat is definiálhatnak.

Láthatóság 2: PPP

Öröklés példa (Docs)

Öröklés korlátozása

Sealed – Nem örökölhető
Kizárás a származtatás alapú módosíthatóságból.
Osztályra és metódusra egyaránt használható.

Singleton – Nem példányosítható
Tervezési modell: csak egyetlen példány létezik, ez szolgál ki mindent.
(static kulcsszóval definiált osztály tulajdonságok és metódusok)

Interfész

Egyirányú kapcsolódási felület strukturális definíciója.
Meghatározza az interfészt megvalósító *programegység* használatának módjait.

Az örökléssel szemben elegendő csak akkor implementálni, ha valóban szükség van rá.

Polimorfizmus

OOP alapelvek

Polimorfizmus 0 – method overload

Method signature: tartalmazza a metódus nevét, paramétereinek típusát és fajtáját (érték vagy hivatkozás)

Az eltérés a metódusok aláírásában van, tehát a metódusok nem keverhetők össze.
A végrehajtandó metódust a paraméterekből a fordító kikövetkezteti.

Method overload példa (Docs)

Polimorfizmus I – method override

A szülő által meghatározott viselkedés módosítása.

Az alacsonyabb absztrakciós szinten specifikus viselkedés szükséges, ezért a származtatott osztály saját viselkedést határoz meg, de nem módosítja az azt előidéző metódus deklarációját.

Override példa (Docs-Print)

Polimorfizmus II – operator override

Egy operátoroknak az osztályra jellemző egyéni viselkedést határozhatunk meg.

Vonatkozik az **index** operátorra is.

Operator overload példa (Docs ==)

Upcast

Típus átalakítás az osztályhierarchia mentén felfelé (a szülők felé).

Az objektummal egy szülője által meghatározott interfészen keresztül kommunikálunk. (Minden *„public”*)

Olyan esetekre, amikor magasabb absztrakciós szinten történő vizsgálat, összevonás szükséges. (Kresz: jármű – személygépkocsi, teherkocsi, busz, motor)

Upcast példa (Docs)

Polimorfizmus III – new

Az override-hoz hasonlóan a szülő által meghatározott viselkedés módosítása, azonban az új viselkedés csak a leszármazottra jellemző.
Amikor az absztrakciós szintek között mozgás funkcióváltozással jár, de az új viselkedés nem változtat a felsőbb szintek (szülők viselkedésén).

New példa (Docs-Print)

Upcast – override, new

Mindkét kulcsszó a leszármazott örökölt viselkedését módosítja, a különbség *UPCAST* esetén, a szülő interfészen keresztül történő kezeléskor jelentkezik.

Örökölt metódus viselkedése a szülő interfészen keresztül hívva

öröklés –a szülő viselkedését tapasztaljuk

override – leszármazott viselkedését tapasztaljuk

new – a szülő viselkedését tapasztaljuk

Interfész példa

Egyirányú kapcsolódási felület strukturális definíciója.
Meghatározza az interfészt megvalósító *programegység* használatának módjait.

Az örökléssel szemben elegendő csak akkor implementálni, ha valóban szükség van rá.

Bár nem az öröklési folyamat része, az *UPCAST* esetén szülőnek számít!!!

Interfész példa (Sorted list; SpaceShuttle)

Kiegészítők

OOP eszközök

Absztrakt osztályok

Olyan osztály, amelynek a pontos működése nem ismert, mert legalább egy absztrakt metódust tartalmaz.

Absztrakt metódus olyan metódus, amelynek nem definiált a metódustörzse.

Nem példányosítható (hiszen a működése nem ismert), származtatáskor az absztrakt metódusokat implementálni kell.

Az öröklési folyamat része!

Abstract példa (Docs-Validator)

Sealed és Singleton

Sealed

Nem örökölhető

Kizárás a származtatás alapú módosíthatóságból.

Osztályra és metódusra egyaránt használható.

Singleton

Nem példányosítható - Tervezési modell

Csak egyetlen példány létezik, ez szolgál ki mindent. A `static` kulcsszóval definiált osztály tulajdonságok és metódusok vagy egy statikus példány alkotja.

Minőség - Kutyaház

Eszközök

- Egységbe zárás
- Öröklés
- Többalakúság
- Öröklés kiegészítése
- **Kohézió – Függetlenítés**
- **Egyszeres felelősség**

Terv

Objektum orientált, olvasható, karbantartható, könnyen újrahasznosítható kód

crosssec SOLUTIONS

Segédelvek

Objektumorientált Programozás

Segédelvek

Cohesion – összetartozás szintje (komponensen belüli függés)

Az osztályok metódusait a közös adatok és (belső) hívások tartják össze.

Egyszeres felelősség elve – SRP – Single Responsibility Principal

Az osztályok metódusait a felelősségi zóna tartja össze.

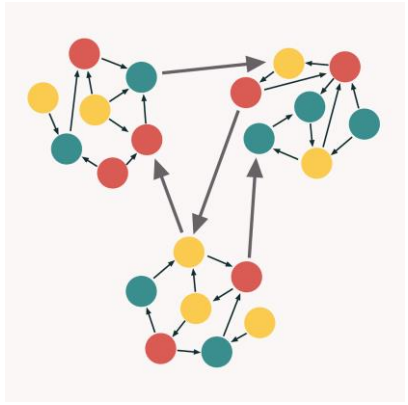
Az egységbezárás tervezésekor tartsuk szem előtt a felelősségi zónákat.

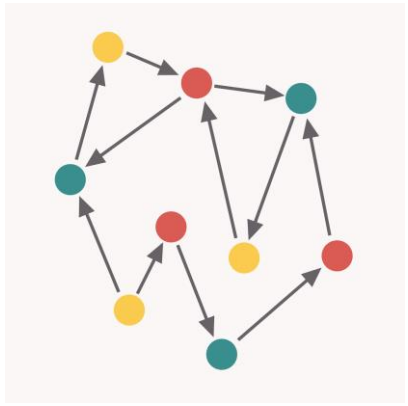
Nem csak az adatokkal és hívásokkal, hanem az „elméleti” működéssel kapcsolatban is.

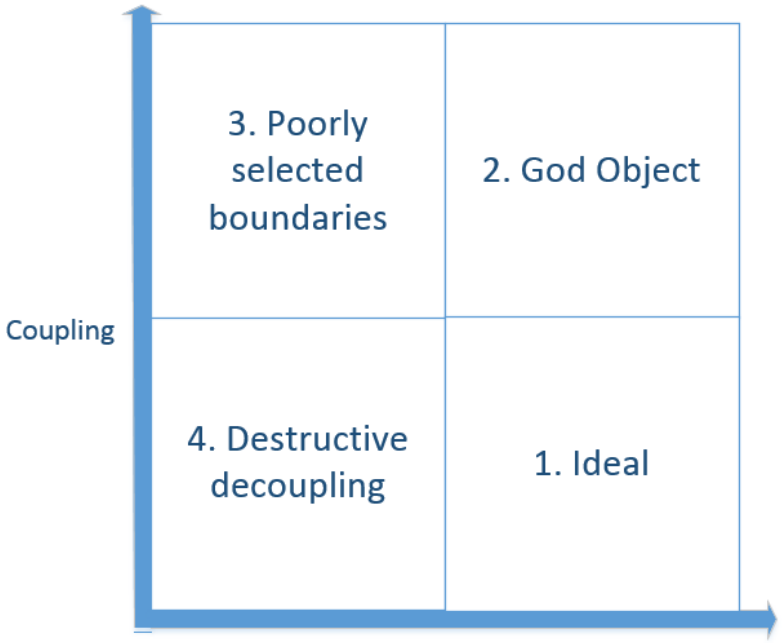
Coupleing – függőség szintje (komponensek közötti függés)

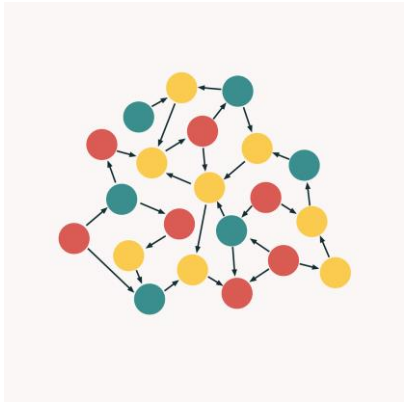
Az osztályok függősége: egy változtatás az osztályhierarchia mekkora részét érinti.

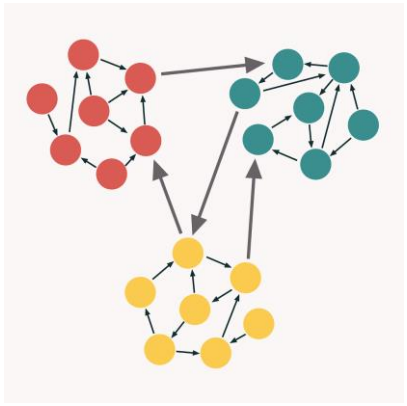
Összetartozás és függőség











Összetartozás és függőség II

Destruktív függőségátlemintés implementáció

```
public class Order                    public class Order
    public Order(IOrderLineFactory factory, IOrderPriceCalculator calculator) {
        _factory = factory;                   public decimal Amount
        _calculator = calculator;
    }                                     get { return _lines.Sum(x => x.Price); }
    public decimal Amount {
        get { return _calculator.CalculateAmount(_lines); }
    }                                 public void AddLine(Product product, decimal amount)
    public void AddLine(IProduct product, decimal price) {
        _lines.Add(_factory.CreateAddOrderLine(product, price, amount));
    }                                 }
}                                 }
```

Dependency Injection

Tervezési modell, melyben egy komponens függőségeit egy másik komponens határozza meg.

Függőség csökkentő eljárás.

Típusai:

- Konstruktor
- Tulajdonság
- Interfész

Összefoglalás

Alapelvek

- Egységbe zárás
  - Konzisztencia, absztrakció
- Öröklés
  - Újrahasznosítás
- Többalakuság – override, new
  - Absztrakciós szint függő viselkedés

Segédelvek

- Öröklés kiegészítés
  - Interface, Abstract, Sealed
- Kohézió
- Egyszeres felelősség
- Függetlenítés (DeCoupling)

Refactoring

Korábban megírt kód újra strukturálása.

Oka

A korábbi munka folyamán kevesebb rendelkezésre álló ismeret.

Funkcionális változtatást nem jelent!

A megírt kód újra strukturálása a funkcionalitás megőrzésével.

Automatikus egységtesztek

Kérdések - Válaszok

Objektumorientált Programozás

Kérdés

Absztrakció: Point_2D

```
private double x = default(double);
public double X { get {return x;} }
```

helyett

```
public double X { get; private set; }
```

is használható?

Kérdés

Sigleton

```
public class Singleton
{
   private static Singleton instance;
   private Singleton() {}

   …

}
```

```
public static Singleton Instance
{
   get
   {
      if (instance == null)instance = new Singleton();
      return instance;
   }
}
```