EKZ

GIT

- Clone
- Fetch
- Pull
- Commit
- Push

COLLECTIONS

- Array

```
//static
int[] intArrayStatic = new int[5];
//dynamic
int [] intArray;
intArray = new int[5];
string[] strArray = new string[5] {"Ronnie", "Jack", "Lori", "Max",
"Tricky"};

string[] strArray = {"Ronnie", "Jack", "Lori", "Max", "Tricky"};

//multidimensional
int[,] numbers = new int[3, 2] { {1, 2}, {3, 4}, {5, 6} };
string[,] names = new string[2, 2] { {"Rosy","Amy"}, {"Peter","Albert"} };

int[,] numbers = new int[,] { {1, 2}, {3, 4}, {5, 6} };
string[,] names = new string[,] { {"Rosy","Amy"}, {"Peter","Albert"} };

int[,] numbers = { {1, 2}, {3, 4}, {5, 6} };
string[,] siblings = { {"Rosy", "Amy"}, {"Peter", "Albert"} };

//multidimensional != jagged array
int[][] numArray = new int[][] {
        new int[] {1,3,5},
        new int[] {2,4,6,8,10}
};
```

Sorting

```
string[] b = { "Bob", "Ivan", "Volodja", "trikytnuk" };
Array.Sort(b);
```

Сортування за ключем

```
//Array.Sort(keys, values);
Tasks[] tasks;
int[] priority ;
Array.Sort(priority, tasks);
```

Сортування частини масиву

```
Array.Sort(array, index, length);
```

Array.Sort(array, index, length);

- Comparer

```
public class myReverserClass : IComparer

{
        int IComparer.Compare( Object x, Object y )
        {
                return( (new CaseInsensitiveComparer()).Compare( y, x ) );
        }
}
```

- Collections
  - ArrayList
  - HashTable
  - SortedList
  - Queue
  - Stack
  - Comparer
  - IEnumerator

LINQ

- **Object Initializations**

```
Invoice i = new Invoice { CustomerId = 123, Name = “Test” };
```

- **Anonymous Types**

```
var i3 = new {CustomerId = 123, Name = “SmithCo” };
var i4 = new {123, “SmithCo”};
```

- **Extension Methods - LINQ**

- Need using `System.Linq`
- Has to be **static**

```
namespace ExtensionMethods
{
        public static class MyExtensions
        {
                public static int WordCount(this String str)
                {
                        return str.Split(new char[] { ' ', '.', '?' },
                        StringSplitOptions.RemoveEmptyEntries).Length;
                }
        }
}
....
using ExtensionMethods;
string s = "Hello Extension Methods";
int i = s.WordCount();
```

Classes

- **const vs readonly**
  const - compile time initialization
  readonly - runtime init

Antipatterns

Code smell - симптоми коду, що вказують на глобальну проблему

- Дублювання коду
- Довгий метод
  Вирішення: розбиття коду по класах та функціях.

```
public void ThisTookTooMuchSpace() {
        So();
        We();
        DividedIt();
        Into();
        Neat();
        Parts();
}
```

- **Великий клас** (god object)
- **Заздрість** (feature envy) - надмірне використання іншого класу
  Вирішення: злиття класів у один.
- **Inappropriate intimacy** - залежність від реалізацій іншого класу
  Вирішення: краща абстракція або реструктуризація класів.

- **Refused request** - ігнорування виконання контракту базового класу
- **Lazy class** - клас, що робить **дуже мало**
- **Excessively long identifiers**
- Зворотня проблєма - **таємничий код**.
- **Надмірне використання літералів**

Забагато коду в одному місці

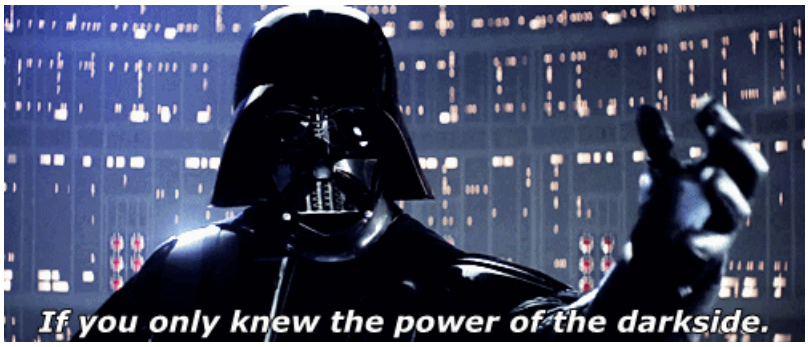
- тяжко читати (**spaghetti code**)
- погане розділення по предметній області

Симптоми:

- *довгий метод*
- *god object*

Actual Antipatterns

Типові помилки у створенні програмних продуктів.



Cycle dependency

- Не є проблемою в одному модулі.
- Є проблемою у різних модулях

Потенційні проблеми:

- **Ефект доміно**: невеликі зміни поширюються на інші модулі
- **Нескінченна рекурсія** та інші збої
- **Memory leaks** через запобігання автоматичному збиранні сміття.

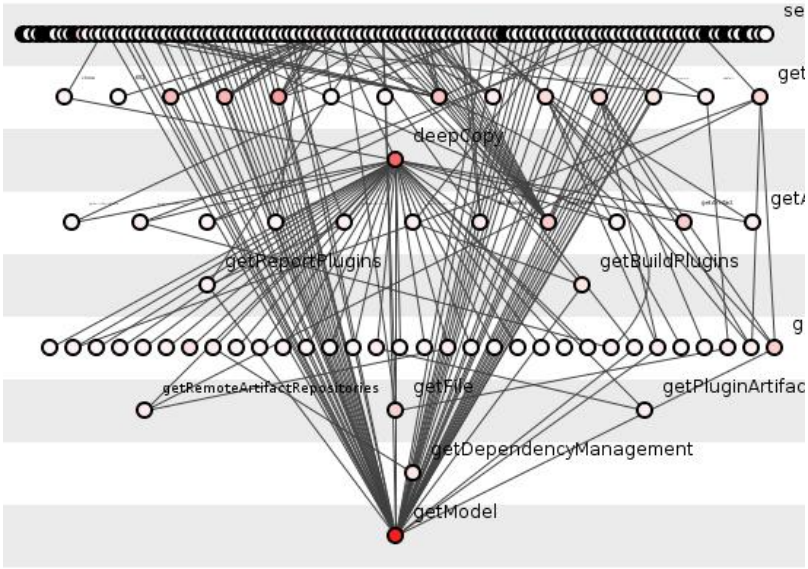
Вирішення:

- Перебудова класів;
- Observer pattern
- Перенесення залежності у абстракцію

God Object

Об'єкт, який дозволяє собі **забагато**.
Ознаки:

- Важко модифікувати.
- Більшість об'єктів мають посилання на нього
- Повинен "знати" всі типи об'єктів.
  Недоліки:
- Волохаті залежності



- Циклічні залежності
  God object знає про всі об’єкти, отже він має
  бути в доступній **Assembly** (яку використовують всі об’єкти).
  Але він робить операції зі всіма об’єктами – отже, ці об’єкти повинні міститись у його **Assembly**, або вище.

При змінах в **God object** є потреба міняти код в безлічі місць, і просто не знаєш з якого боку взятись.

А при спробі **порефакторити** усе падає, через сильні залежності.

Усунення: розбиття функціоналу на відповідні класи.

Adam Object

Універсальний базовий об'єкт.
God Object - залежність використання.
Adam Object - структурна залежність.
Проблєми:

- Зміна у методі Адама призводить до зміни всіх нащадків.
- Негнучка система класів.
- Абстрактні методи Адама мають бути переписані у кожному нащадку.
- Якщо Адам - об'єкт, неможливо наслідувати від іншого об'єкту.
- Приклад Адама - **Шаблонний метод**.
  Рішення:
- Крихкі методи зробити **віртуальними**.
- Змінні робити **приватними** та перевизначати їх у нащадках.
- Винести частину у інший **класи/інтерфейси**.

Паблік Морозов

Клас-нащадок (exposes), що видає дані класу-предка.

- Клас предок має приховані поля та методи.
- Клас нащадок тим чи іншим чином робить ці поля і методи доступними для всіх (public полями та методами)

Мінуси:

- Порушення інкапсуляції.
- Порушення безпеки.

Object Orgy

Спричинене масовим використанням `public`. Таким чином, об'єкти недостатньо **інкапсульовані**.

Недоліки:

- Код стає нечитабельним.
- Абстракція стає неефективною.
- Spaghetti code.

BaseBean

Антипатерн Base Bean виникає, коли конкретний доменний клас успадковує утилітарний клас, тому що він хоче використовувати утилітарні методи цього утилітарного класу. Це також часто називають успадкуванням для реалізації.

Успадкування від утилітарного класу лише для того, щоб мати можливість використовувати його методи, є поганим способом використання успадкування і

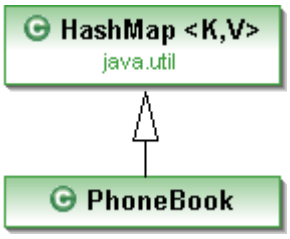
порушує принцип підстановки Ліскова. Це не має сенсу з точки зору семантики домену і тому може бути потенційно заплутаним для інших розробників.

Недоліки:

- Доменний клас покладається на внутрішні компоненти утилітарного класу, які можуть бути поза контролем розробника. Коли змінюється клас утиліти, це впливає на клас домену. Загалом, це зменшує зручність супроводження системи.
- Base Bean призводить до зниження зрозумілості, оскільки спосіб використання успадкування не має сенсу з точки зору домену, який моделюється.
- Ця практика також плутає значення доменного класу і забруднює його методами, які насправді не є частиною концепції, яку він представляє. Таким чином, доменний клас більше не має єдиної відповідальності, оскільки він також перебирає на себе функціональність утилітарного класу.
- Доменний клас може не мати наміру виконувати контракт утилітарного класу.

Вирішення:
Замість того, щоб успадковувати від утилітарного класу, конкретний доменний клас повинен просто делегувати утилітарному класу виклик його методів. Це відповідає максимі проектування "Надавайте перевагу композиції, а не успадкуванню".

BAD PROGRAMMER



GOOD PROGRAMMER 

CallSuper

Call Super - це проблема в об'єктно-орієнтованих фреймворках під час створення підкласу для інтеграції з фреймворком. API повинен виконувати необхідні кроки. Використання патерну **Template Method** може вирішити цю проблему.

Приклади коду:

Оригінал:

```
public class EventHandler ...
  public void handle (BankingEvent e) {
```

```
    housekeeping(e);
  }
public class TransferEventHandler extends EventHandler...
  public void handle(BankingEvent e) {
    super.handle(e);
    initiateTransfer(e);
  }
```

Покращений варіант:

```
public class EventHandler ...
  public void handle (BankingEvent e) {
    housekeeping(e);
    doHandle(e);
  }
  protected void doHandle(BankingEvent e) {
  }
public class TransferEventHandler extends EventHandler ...
  protected void doHandle(BankingEvent e) {
    initiateTransfer(e);
  }
```

Circle-Ellipse Problem

Може виникнути при використанні поліморфізму підтипів при моделюванні об’єктів.

- Який клас має бути базовий – квадрат чи прямокутник?
  Недоліки:
- Порушення Liskov substitution principle.
- Нащадок має підтримувати абсурдний для себе метод базового класу.
  Розв'язок:
- Перебудова класів
- Повертання булевого значення
- Immutable – повертання нового об'єкту, як результат (а не операції над поточним)

Object Cesspool

- Пастка шаблону Object Pool.
- Перевикористання об'єктів, що знаходяться у непридатному до використання стані.

Poltergeist

- Об'єкт, що живе **мало часу** та засмічує пам'ять.
- Виконує **мінімальне** значення.

Sequential Coupling

- Методи класу можуть бути викликані лише у певному порядку.
- Система не функціонуватиме при зміні порядку.

Yo-Yo Problem

Проблеми

- Глибокі ієрархії
- Підкласифікація для повторного використання коду
- Читабельність
- Висока зв'язність

Рішення

- Надавайте перевагу композиції над успадкуванням.
- Рефакторинг глибоких ієрархій.

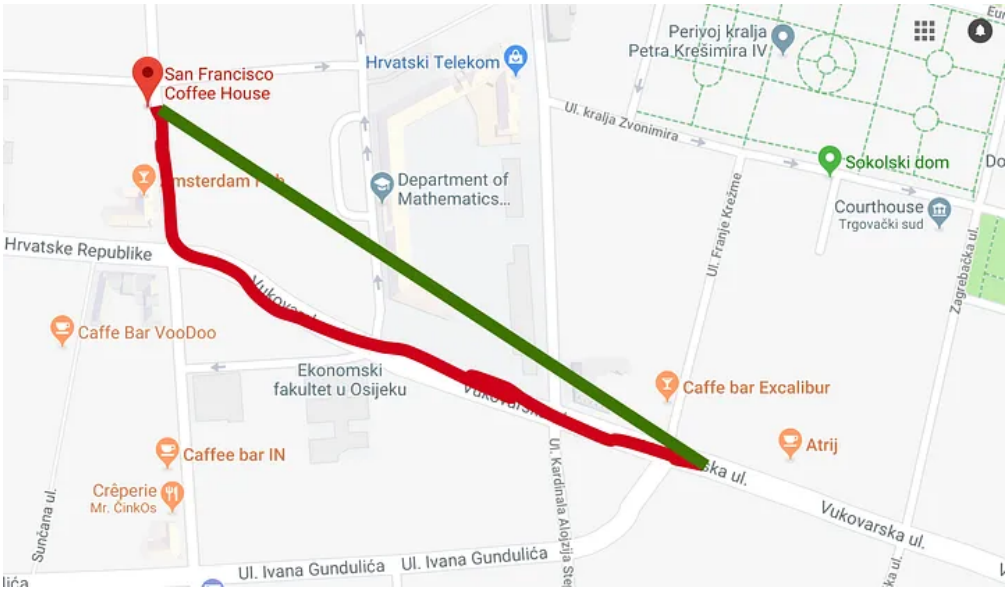
Singletonitis

Зловживання синглтонами.
Проблєми:

- Збирач сміття не видаляє ці об’єкти аж до завершення програми
- При розширенні класу приходиться відмовлятись від цього шаблону (наприклад, вам потрібно різний такий об’єкт для різних вікон)

Accidental Complexity

Надлишкова складність, яка виникає в комп'ютерних програмах або в процесі їх розробки (програмування)



Action At a Distance

Поширена помилка, в якій поведінка в одній з частин програми змінюється в залежності від операції в іншій частині програми. Важко або неможливо визначити, що відбувається (ознака - купа перевірок правильності).

Усунення проблеми:

- Уникнення глобальних змінних
- Змінювати дані лише у локальних межах (своїх класах чи своїй частині програми)
- Шаблон “Стан”

Blind Faith

- no unit tests
- no try-catch
- no bitches

Boat Anchor

Збереження більше не використовуваної частини системи

Busy Spin (Активне Очікування)

Використання ресурсів ЦП під час очікування на подію через постійну перевірку.

Очевидним рішенням є система повідомлень.

Caching Failure

Не скидання кешу помилки після її оброблення.

Наприклад, у браузері є кеш, якщо помилка закешувалась, то на повторне введення адреси треба не використовувати кеш, а обновити його

The Diaper Pattern Stinks

Ігнорування помилки або передача на обробку вище.

```
try {
        ...
}
catch {
        // dont care didnt ask
}
```

Проблєми:

- нагромадження помилок
- неможливість відслідкування помилки

Hiding Exceptions (Хования Помилок)

```
try {
        ...
}
catch {
        throw new Exception("Error :) Won't tell you why though :)")
}
```

Coding By Exception

- Нагромадження перевірок і виняткових ситуацій
- Для кожного випадку призначається свій клас `Exception`.

Cargo Cult Programming

Нецільове використання патернів проектування чи архітектури.

Hard Code

Жорстка прив'язка до даних у коді

Soft Code – винесення непотрібних речей у config.

Magic Strings

Використання стрічок для передачі даних замість окремих типів

Ковертування у стрічки та з них стає незручним у підтримці та джерелом помилок.

Magic Numbers

Welp, magic numbers і guess

```
1
2  float FastInvSqrt(float x) {
3    float xhalf = 0.5f * x;
4
5    int i = *(int*)&x;            // evil floating point bit level hacking
6    i = 0x5f3759df - (i >> 1);    // what the fuck?
7
8    x = *(float*)&i;
9    x = x*(1.5f-(xhalf*x*x));     //1st iteration
10   //x = x*(1.5f-(xhalf*x*x));  // 2nd iteration, this can be removed
11
12   return x;
13 }
14
```

Lava Flow

Збереження небажаного (зайвого або низькоякісного) коду через те, що його вилучення занадто дорого або буде мати непередбачувані наслідки.

Методологічні антипатерни

- Copy-paste – при зміні одної з частин коду легко забути про інші.

- Defactoring – Заміна функціональості документацією.
- Silver bullet – Застосування улюбленого інструменту (шаблону) до всіх без виключення проблем.
- Improbability factor – Припущення про неможливість того, що спрацює відома помилка.
- Programming by accident – полягає у випадковій зміні малих кусків коду і перевірки, чи програма запрацювала.
- Reinventing the wheel
- Reinventing the square wheel

Конфігураційні антипатерни

- Dependency hell. Проблеми з версіями потрібних продуктів. Ця проблема виникла в ранніх версіях Microsoft Windows.

90-90 rule

> **Quote**
>
> "The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time."
> —Tom Cargill