Vzor Singleton

Potrebujeme presne jeden objekt

- Existujú triedy objektov také, že v aplikácii potrebujeme práve jeden objekt danej triedy:
  - Preferences
  - Registry settings
  - Database connection
  - Printer driver
- V niektorých prípadoch by vytvorenie viacerých objektov danej triedy mohlo spôsobiť problémy
  - Napríklad objekty náročné na pamäť a iné systémové zdroje

Prečo návrhový vzor?

- Problém jedného objektu sa zrejme dá riešiť mnohými spôsobmi:
  - Globálne premenné
  - Static variables
  - Singleton sa stal konvenciou a štandardom pre riešenie tohoto problému
  - Výhoda napríklad oproti globálnej premennej:
  - Globálnu premennú musím nainicializovať pri spustení aplikácie. Singleton vtedy, keď ho potrebujem

Problém

- Ako zariadiť, aby nebolo možné vytvoriť viac než jeden objekt nejakej triedy?
- ...skúste niečo vymyslieť!

Ako vytvoríme objekt?

```
new MyObject();
```

Takýchto volaní konštruktora môžeme mať samozrejme ľubovoľne veľa.

Ak je MyObject public class, môže takýto objekt vytvoriť akýkoľvek iný objekt

Čo takáto konštrukcia?

```
public MyClass{
        private MyClass()}
```

Toto je z hľadiska jazyka legálna konštrukcia, avšak zdanlivo nedáva zmysel:

Iba objekty typu MyClass môžu volať konštruktor MyClass. To je nám však na nič, lebo objekt typu MyClass nemá ako vzniknúť

Ešte iná konštrukcia

```
public MyClass{
    private MyClass() {}
    public static MyClass getInstace(){
        return new MyClass();
    }
```

Static vlastnosti a metódy

- Static vlastnosti a metódy patria triede. Je možné proužívať ich aj vtedy, keď neexistuje žiaden objekt danej triedy
- Obsahujú dáta a správanie, ktoré sú nezávislé od konkrétneho objektu
- Static members sú inicializované systémom (.NET framework, JVM) pred tým, než sú prvýkrát použité

Static:príklad

```
public class Automobile
{
    public static int NumberOfWheels = 4;
    public static int SizeOfGasTank
    {
        get
        {
            return 15;
        }
    }
    public static void Drive() { }
    public static event EventType RunOutOfGas;

    //other non-static fields and properties...
}
```

Static trieda

- static môže byť aj trieda. Znamená to, že obsahuje iba static members.
- Nie je možné vytvárať inštancie static tried

Späť k singletonu

```
public MyClass{
    private MyClass() {}
    public static MyClass getInstace(){
        return new MyClass();
    }
```

Vytváranie objektov triedy MyClass

Pri použití uvedenej konštrukcie sa nové objekty triedy MyClass budú vytvárať jednoducho:

```
MyClass.getInstance();
```

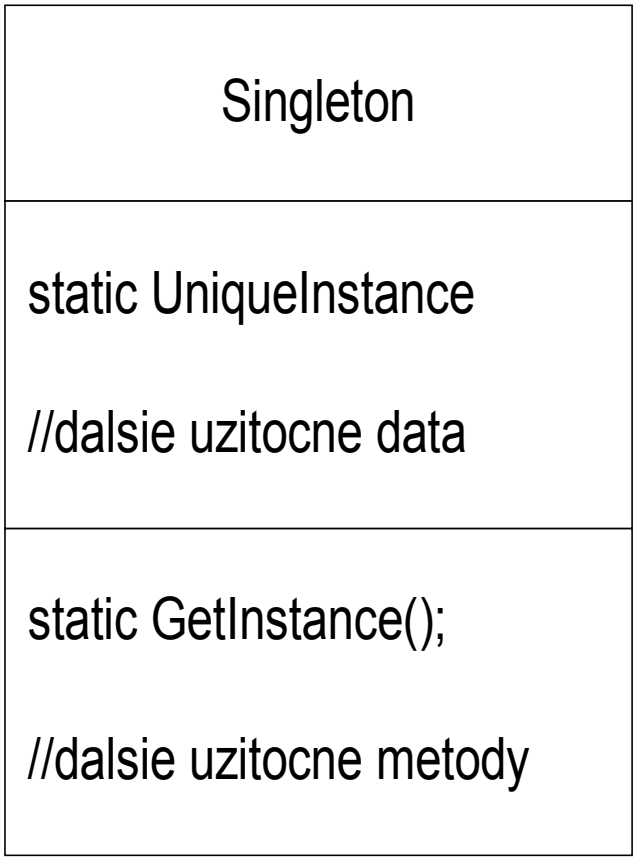
Posledná úprava konštrukcie

```
public Singleton{
   private static Singleton UniqueInstance;

   private Singleton() {}

   public static Singleton getInstace(){
       if (UniqueInstance == null) {
          UniqueInstance = new Singleton();
       }
       return Singleton();
   }
   //Dalsie uzitocne metody
}
```

Singleton: class diagram



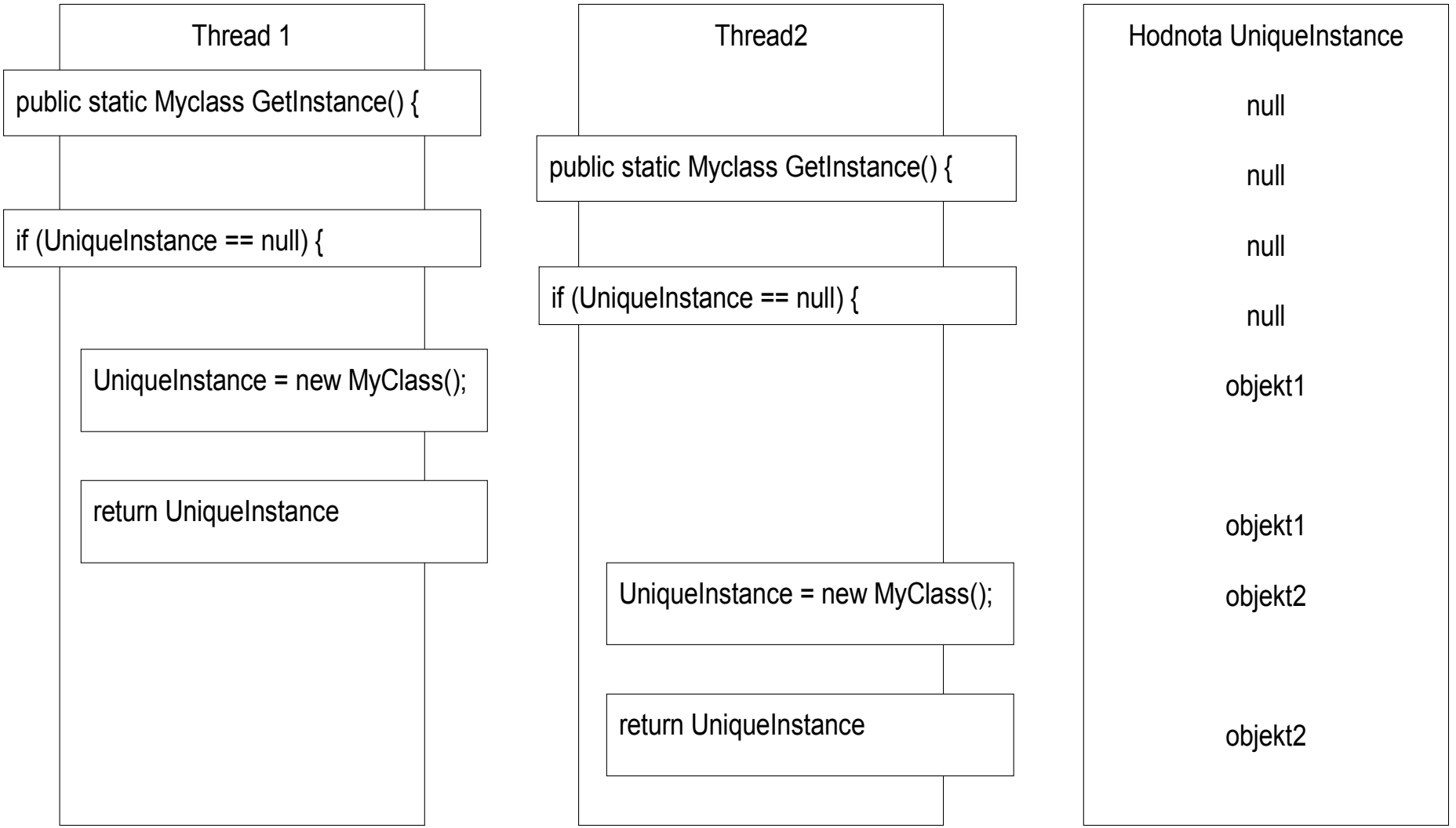
```
class MSSqlConnection {
    private static MSSqlConnection uniqueInstance;
    private string connectieString = @"Data Source=.\SQLEXPRESS;Database...“
    private SqlConnection connectie;

    private MSSqlConnection() { }
    public static MSSqlConnection getInstance()  {
        if (uniqueInstance == null) {
            uniqueInstance = new MSSqlConnection();
        }
        return uniqueInstance;
    }
    public void Pripoj(){
        connectie = new SqlConnection(connectieString);
        connectie.Open();
    }
    public SqlConnection getConnectie()  {
        return connectie;
    }
    public void Odpoj() {
        connectie.Close();
    }
  }
}
```

Prečo chceme iba jedno pripojenie?

- Každá aplikácia potrebuje presne jedno pripojenie
- Správa pripojení sa typicky zanedbáva. Bežná aplikácia má otvorené desiatky pripojení
- Počet pripojení k databázovému serveru môže byť obmedzený na strane servera
- Vysoký počet otvorených pripojení môže spôsobiť problémy s výkonnosťou

Potenciálny problém: multithreading



Riešenie problému

Jedným z možných riešení tohoto problému je „proaktívne“ vytvorenie inštancie:

```
public class Singleton {
   private static Singleton UniqueInstance
    = new Singleton();

   private Singleton() {}

public static Singleton getInstance() {
      return UniqueInstance;
   }
}
```

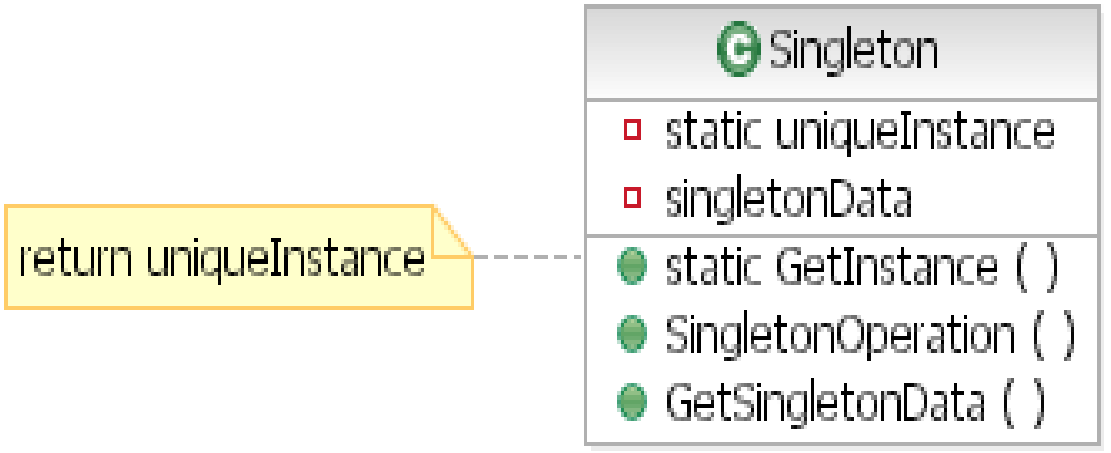
Singleton podľa GoF

- Názov: Singleton
- Klasifikácia: Object creational
- Zámer: Zaistiť, aby trieda mala iba jednu inštanciu a poskytnúť globálny access point k tejto inštancii
- Motivácia
  - Potrebujeme také triedy, ktoré budú mať iba jednu inštanciu
  - Príklad: printer spooler, file system,...
  - Najlepšie je vytvoriť triedu, ktorá sama bude dohliadať na vytváranie inštancií

Singleton podľa GOF II.

- Aplikovateľnosť: Singleton sa používa tam, kde:
  - Musí existovať iba jedna inštancia triedy a táto musí byť prístupná z jedného bodu
  - Potrebujeme jednu inštanciu rozšíriteľnú pomocou dedičnosti. Klienti budú pristupovať k rozšírenej inštancii bez toho, aby modifikovali jej kód (???)

Singleton podľa GoF: Štruktúra



Singleton podľa GoF: Účastníci

- Singleton:
  - definuje operáciu GetInstance(), pomocou ktorej môžu klienti pristupovať k jeho jedinej inštancii
  - môže byť zodpovedný za vytvorenie svojej jedinej inštancie

Singleton podľa GoF: Spolupráca

- Klienti pristupujú k inštancii singletonu výlučne pomocou operácie GetInstance()

Singleton podľa GoF: Dôsledky

- Singleton má nasledujúce výhody
  - Kontrolovaný prístup k jedinej inštancii
  - Redukovaný namespace: Singleton je vylepšním oproti globálnym premenným. Zabraňuje znečisteniu name space globálnymi premennými obsahujúcimi jedinečné inštancie
  - Umožňuje prepísať operácie a reprezentácie pomocou dedičnosti (???)
  - Umožňuje povolenie viacerých inštancií (ak sa rozhodneme, že potrebujeme viac inštancií, je táto úprava jednoduchá a izolovaná)

Singleton podľa GoF: Implementácia a ukážka kódu

- Vysvetlenie a príklady implementácie pomocou jazyka C++

Singleton podľa GoF: Známe použitia

- Smalltalk-80: vzťah medzi triedami a ich meta-triedami (metatriedy obsahujú definície tried)
- InterViews user interface toolkit: Prístup k jedinečným inštanciám tried Session a WidgetKit

Singleton podľa GoF: príbuzné vzory

- Singleton sa využíva pri realizácii niektorých iných vzorov (Abstract Factory, Builder, Prototype)