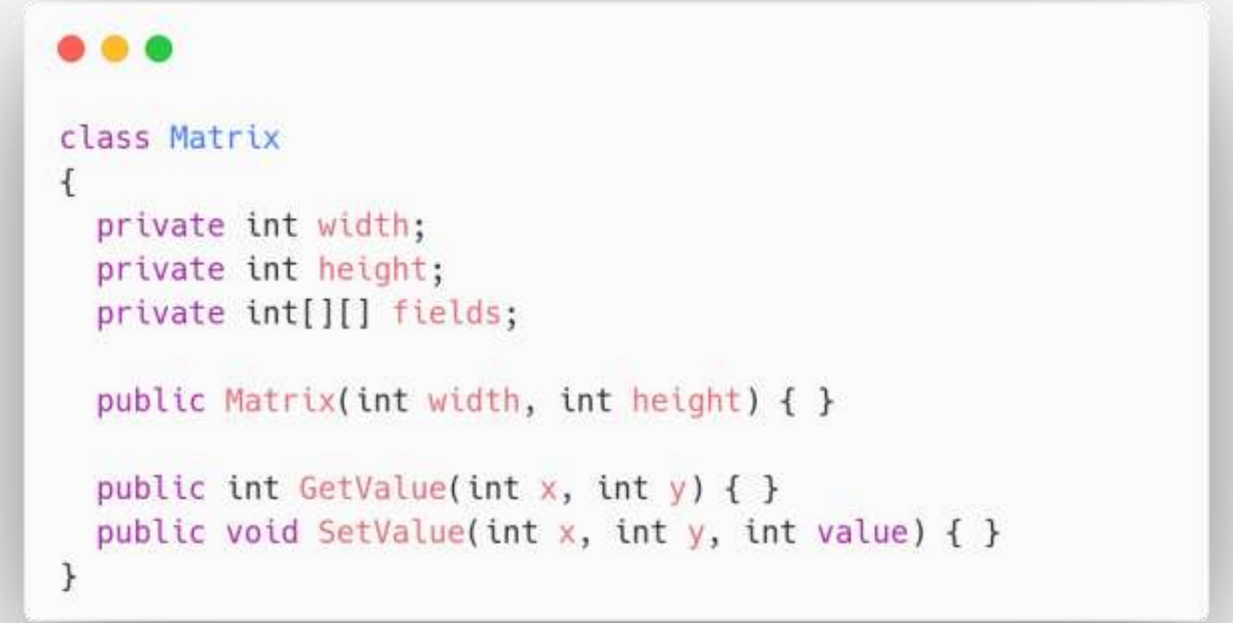
Anti-patterns

Definice

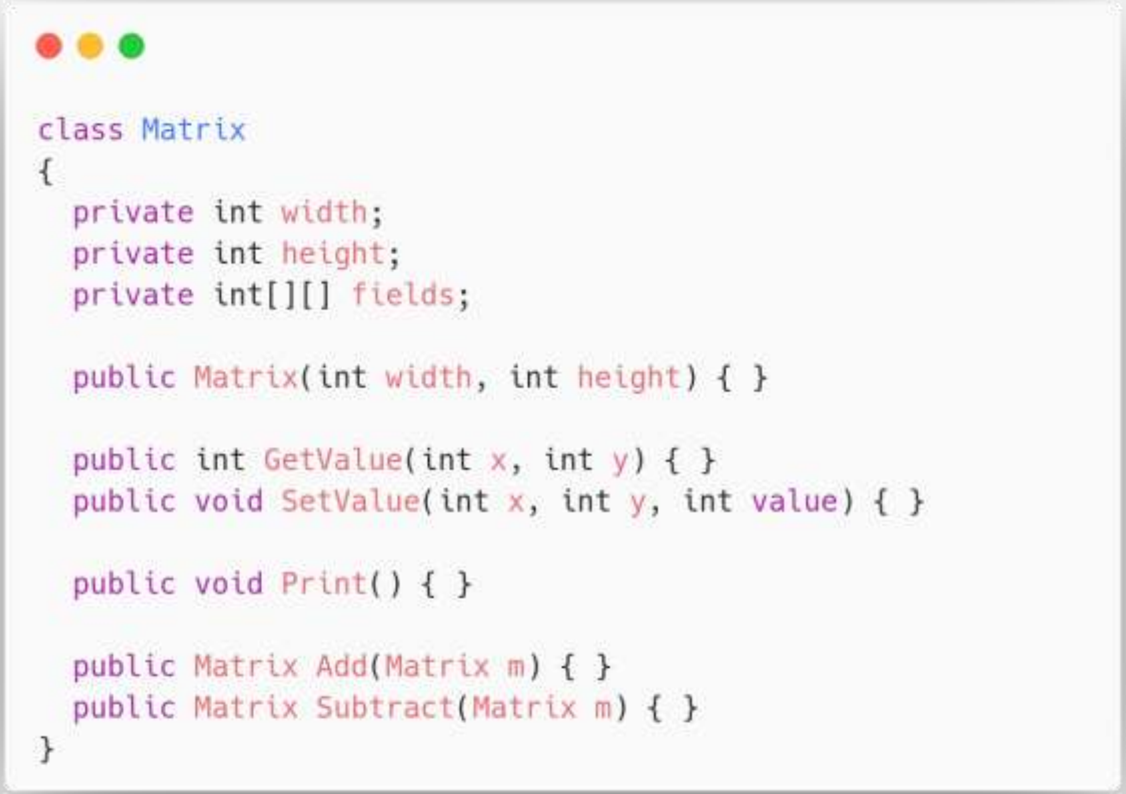
- Anti-pattern je často používané řešení opakujícího se problému, jehož negativní důsledky převažující jeho užitečnost a pro nějž existuje vhodnější alternativa.

Maticová kalkulačka



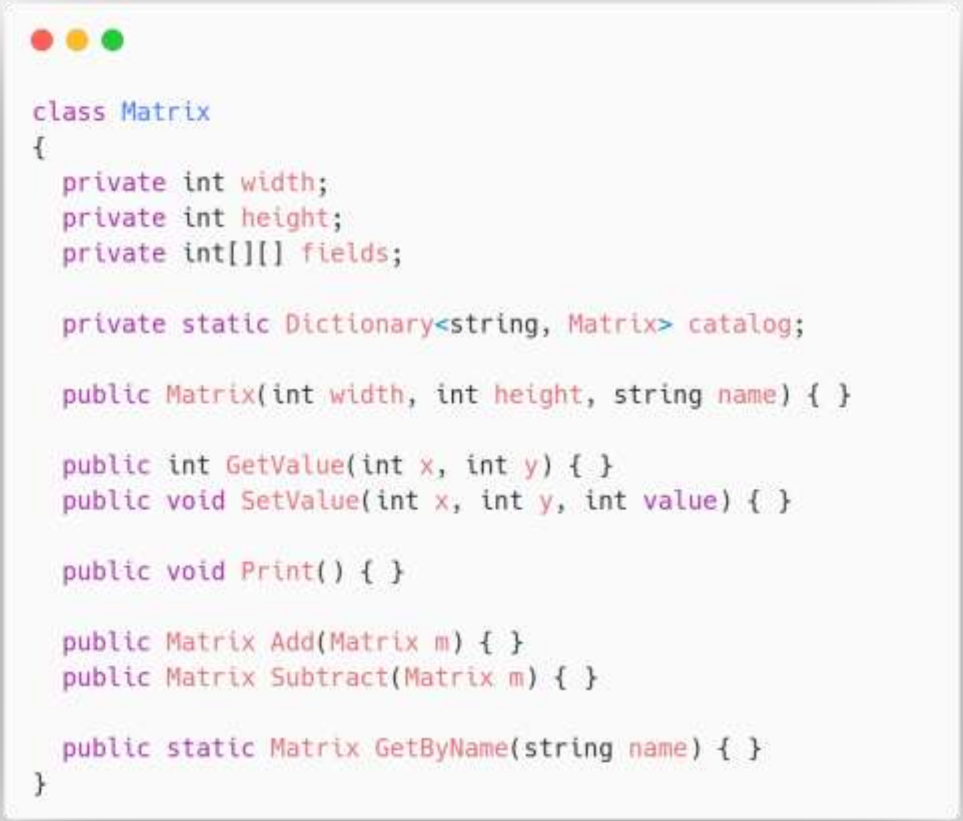
```
class Matrix
{
  private int width;
  private int height;
  private int[][] fields;

  public Matrix(int width, int height) { }

  public int GetValue(int x, int y) { }
  public void SetValue(int x, int y, int value) { }
}
```

Maticová kalkulačka...



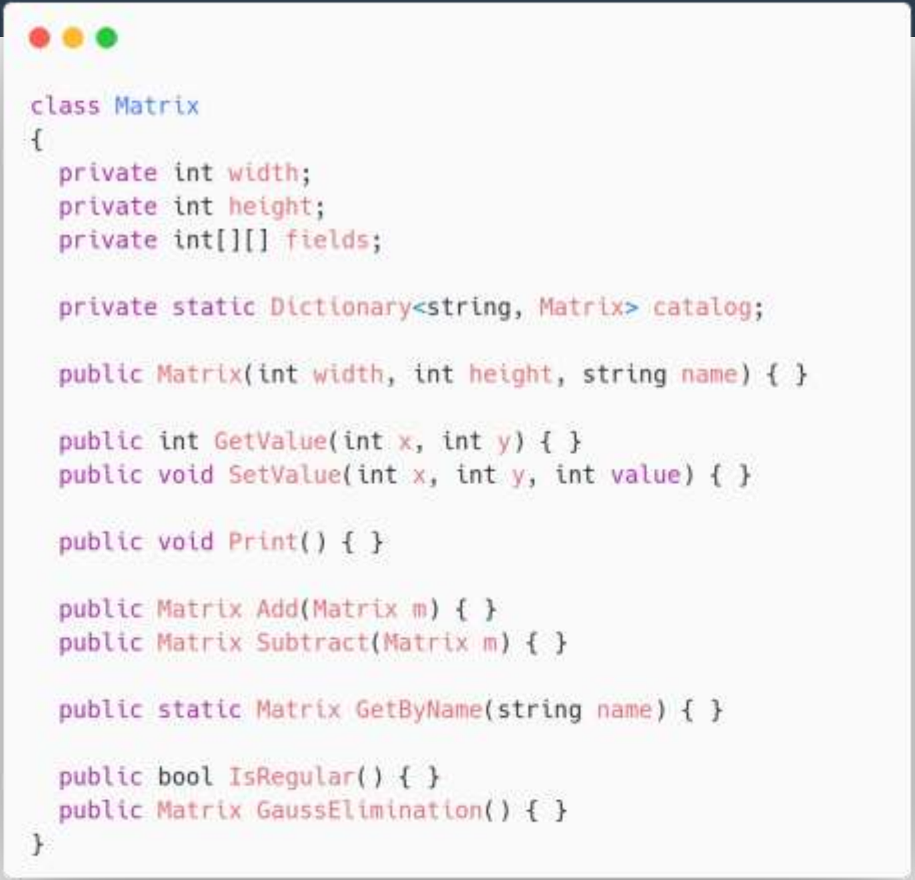
```
class Matrix
{
  private int width;
  private int height;
  private int[][] fields;

  public Matrix(int width, int height) { }

  public int GetValue(int x, int y) { }
  public void SetValue(int x, int y, int value) { }

  public void Print() { }

  public Matrix Add(Matrix m) { }
  public Matrix Subtract(Matrix m) { }
}
```

Maticová kalkulačka?



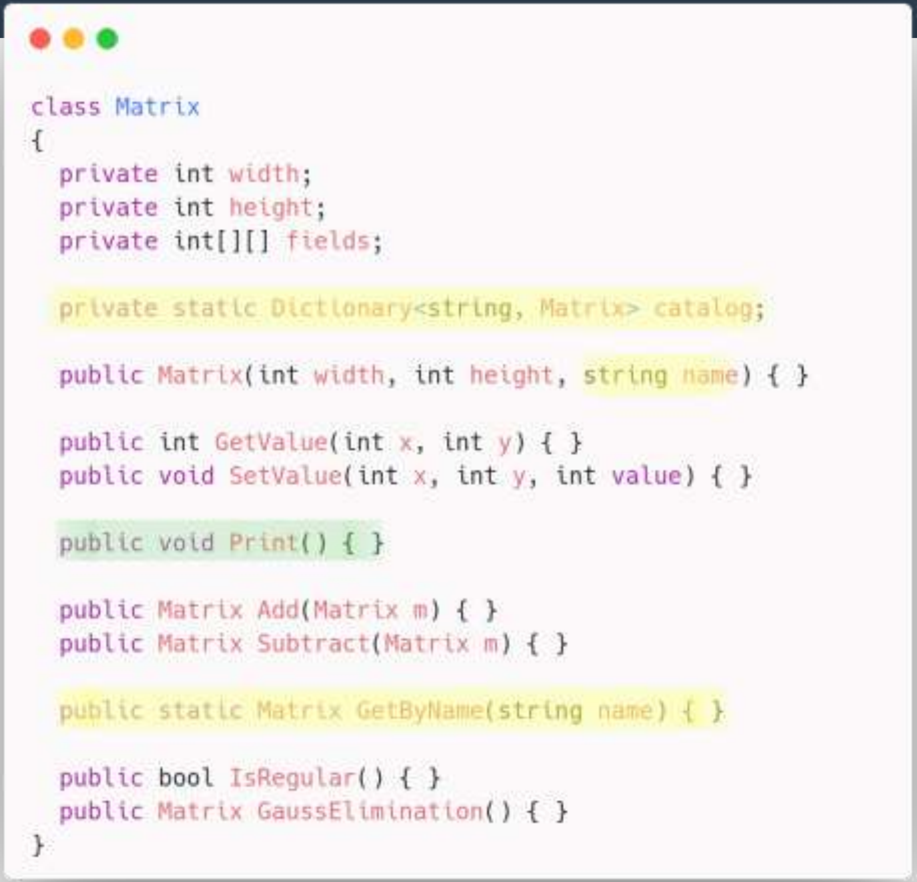
```
class Matrix
{
  private int width;
  private int height;
  private int[][] fields;

  private static Dictionary<string, Matrix> catalog;

  public Matrix(int width, int height, string name) { }

  public int GetValue(int x, int y) { }
  public void SetValue(int x, int y, int value) { }

  public void Print() { }

  public Matrix Add(Matrix m) { }
  public Matrix Subtract(Matrix m) { }

  public static Matrix GetByName(string name) { }
}
```

Maticová kalkulačka!



```
class Matrix
{
  private int width;
  private int height;
  private int[][] fields;

  private static Dictionary<string, Matrix> catalog;

  public Matrix(int width, int height, string name) { }

  public int GetValue(int x, int y) { }
  public void SetValue(int x, int y, int value) { }

  public void Print() { }

  public Matrix Add(Matrix m) { }
  public Matrix Subtract(Matrix m) { }

  public static Matrix GetByName(string name) { }

  public bool IsRegular() { }
  public Matrix GaussElimination() { }
}
```

Blob

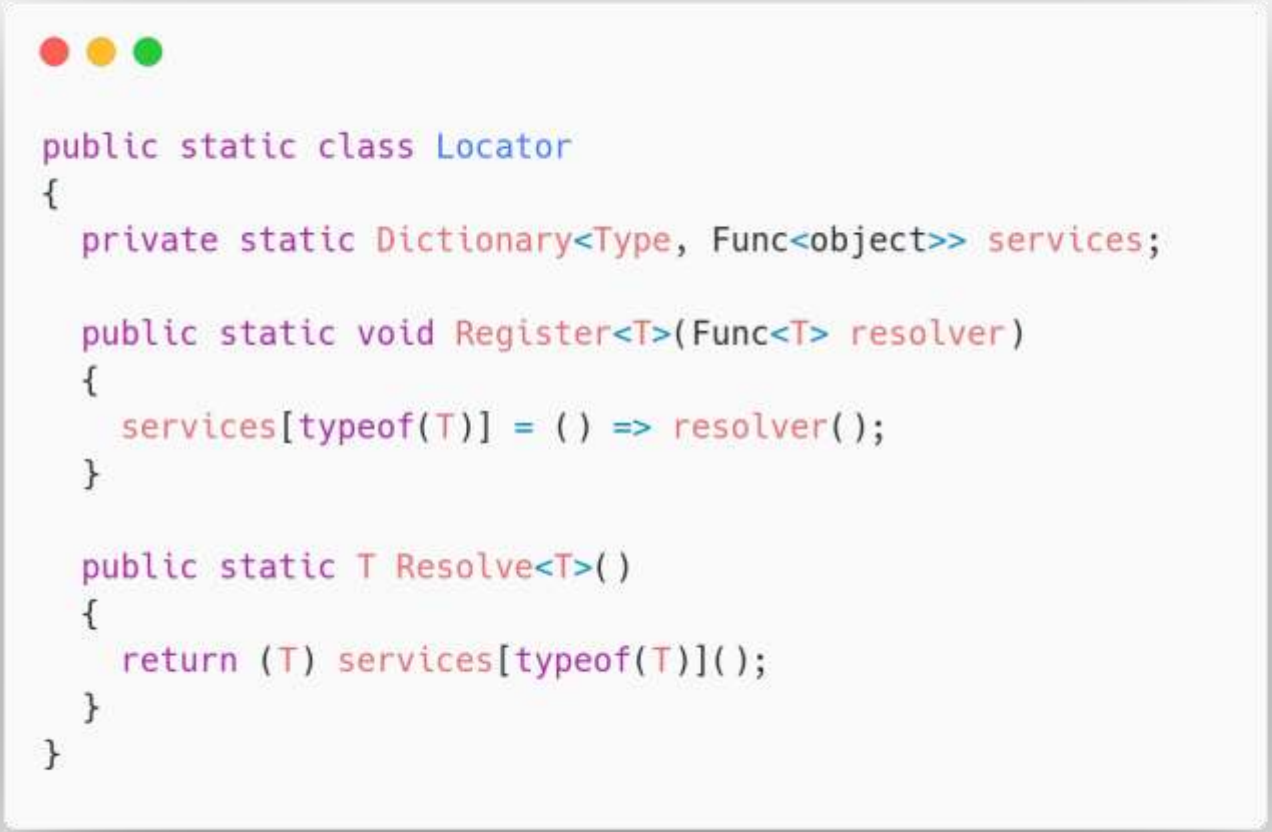


```
class Matrix
{
  private int width;
  private int height;
  private int[][] fields;

  private static Dictionary<string, Matrix> catalog;

  public Matrix(int width, int height, string name) { }

  public int GetValue(int x, int y) { }
  public void SetValue(int x, int y, int value) { }

  public void Print() { }

  public Matrix Add(Matrix m) { }
  public Matrix Subtract(Matrix m) { }

  public static Matrix GetByName(string name) { }

  public bool IsRegular() { }
  public Matrix GaussElimination() { }
}
```

Service Locator

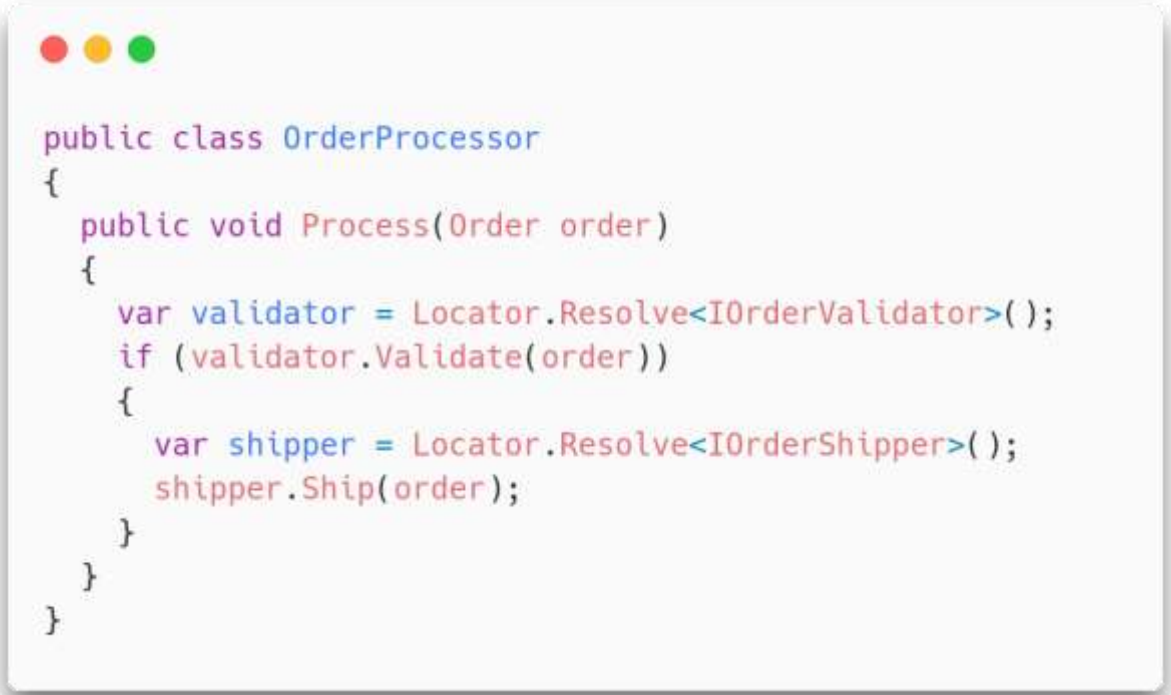
- Projekt složený ze spousty komponent
- Komponenty potřebují konzumovat různé závislosti
- Potřebujeme závislosti nějak dostat do komponent
- Chceme se vyhnout těsným vazbám

Jak vypadá Locator

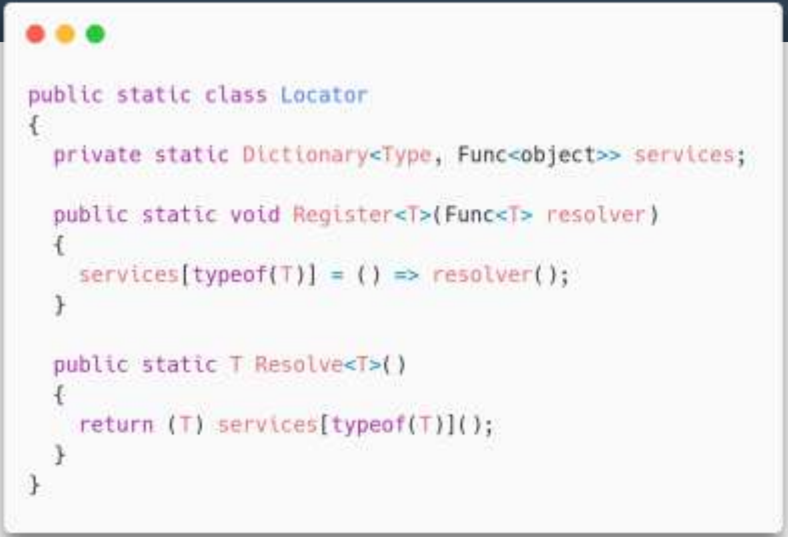


```
public static class Locator
{
  private static Dictionary<Type, Func<object>> services;

  public static void Register<T>(Func<T> resolver)
  {
    services[typeof(T)] = () => resolver();
  }

  public static T Resolve<T>()
  {
    return (T) services[typeof(T)]();
  }
}
```

Příklad — OrderProcessor

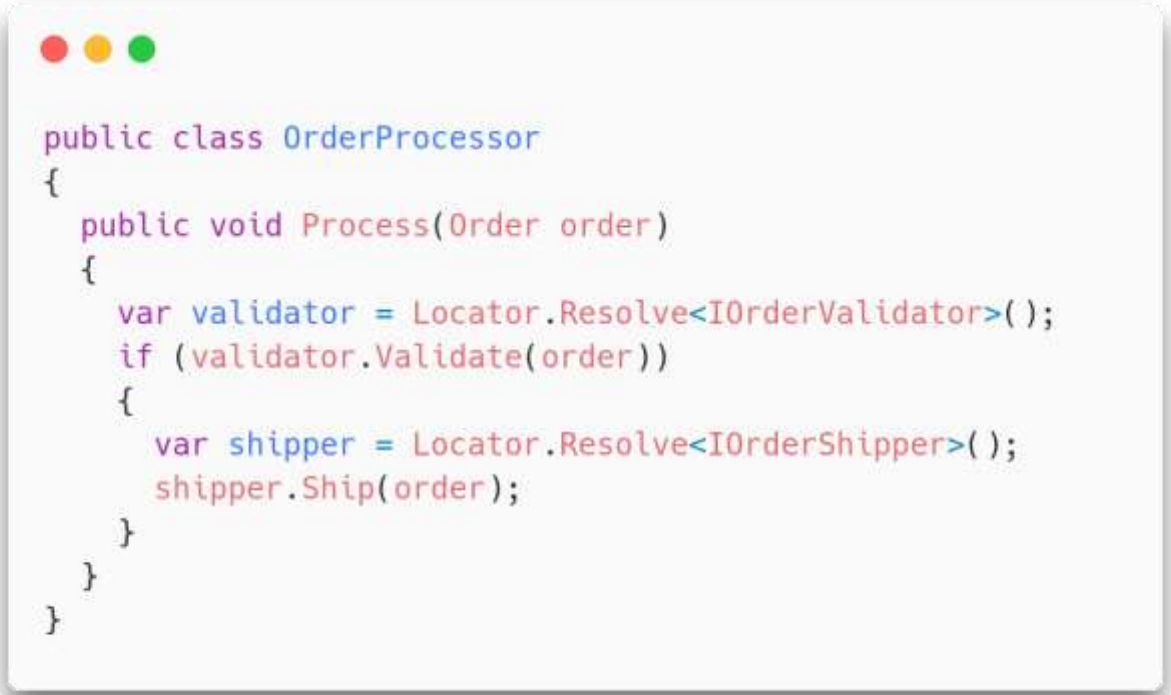


```
public class OrderProcessor
{
  public void Process(Order order)
  {
    var validator = Locator.Resolve<IOrderValidator>();
    if (validator.Validate(order))
    {
      var shipper = Locator.Resolve<IOrderShipper>();
      shipper.Ship(order);
    }
  }
}
```

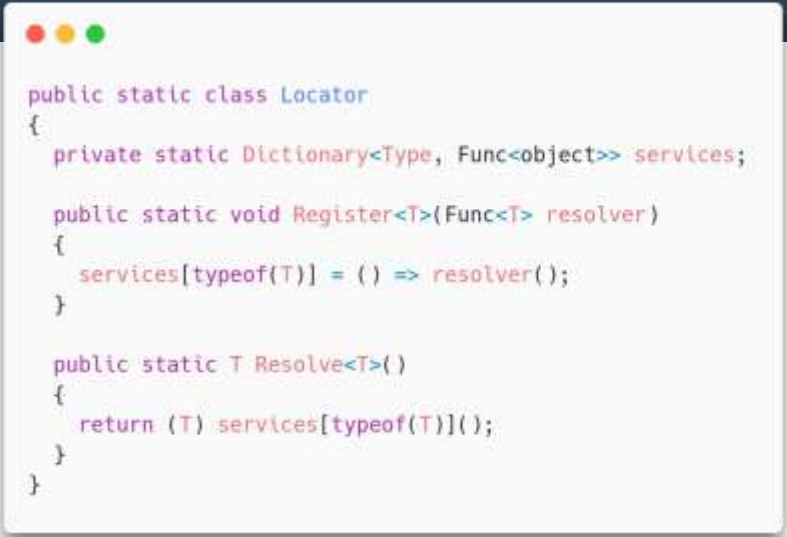


```
public static class Locator
{
  private static Dictionary<Type, Func<object>> services;

  public static void Register<T>(Func<T> resolver)
  {
    services[typeof(T)] = () => resolver();
  }

  public static T Resolve<T>()
  {
    return (T) services[typeof(T)]();
  }
}
```

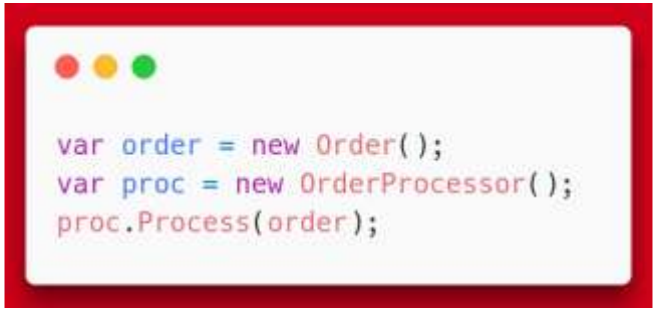
Problém #1 — API



```
public class OrderProcessor
{
  public void Process(Order order)
  {
    var validator = Locator.Resolve<IOrderValidator>();
    if (validator.Validate(order))
    {
      var shipper = Locator.Resolve<IOrderShipper>();
      shipper.Ship(order);
    }
  }
}
```

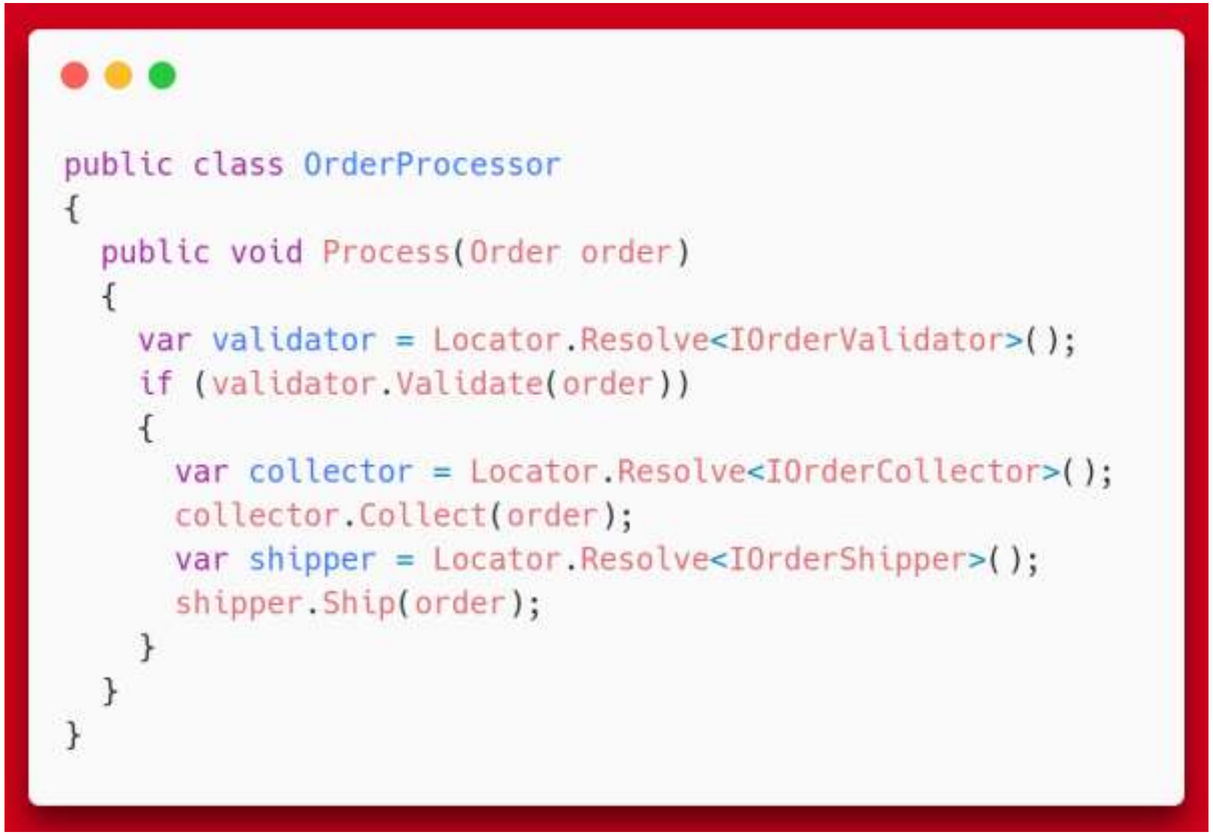


```
public static class Locator
{
  private static Dictionary<Type, Func<object>> services;

  public static void Register<T>(Func<T> resolver)
  {
    services[typeof(T)] = () => resolver();
  }

  public static T Resolve<T>()
  {
    return (T) services[typeof(T)]();
  }
}
```

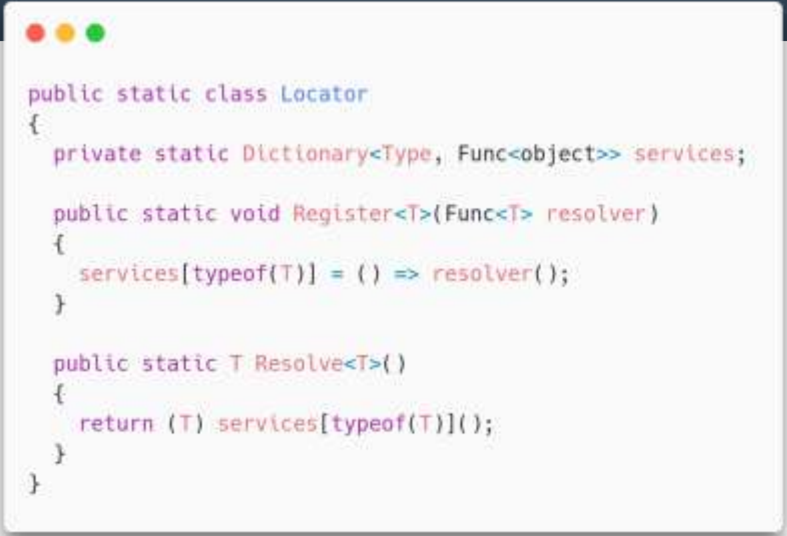


```
var order = new Order();
var proc = new OrderProcessor();
proc.Process(order);
```

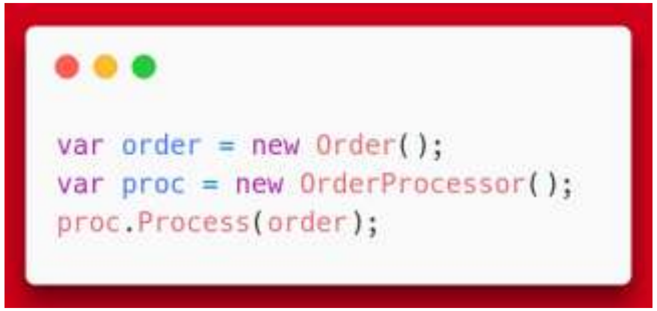
Problém #2 — Údržba



```
public class OrderProcessor
{
  public void Process(Order order)
  {
    var validator = Locator.Resolve<IOrderValidator>();
    if (validator.Validate(order))
    {
      var collector = Locator.Resolve<IOrderCollector>();
      collector.Collect(order);
      var shipper = Locator.Resolve<IOrderShipper>();
      shipper.Ship(order);
    }
  }
}
```



```
public static class Locator
{
  private static Dictionary<Type, Func<object>> services;

  public static void Register<T>(Func<T> resolver)
  {
    services[typeof(T)] = () => resolver();
  }

  public static T Resolve<T>()
  {
    return (T) services[typeof(T)]();
  }
}
```



```
var order = new Order();
var proc = new OrderProcessor();
proc.Process(order);
```

Co s tím?

Jak z kola ven

- Rozumět existujícím návrhovým vzorům
- **Přemýšlet nad návrhem!**
- Dodržovat principy dobrého designu
  - **SOLID**
  - Encapsulation
  - Loose coupling
  - POLA, ...

Literatura

- Brown, Malveau, McCormick, Mowbray (1998) **AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis.** ISBN 978-0-471-19713-3
- Fowler (1999) **Refactoring: Improving the Design of Existing Code.** ISBN 0-201-48567-2.
- Seemann (2011) **Dependency Injection in .NET** ISBN 9350042371