Fundamentals: The Dependency Inversion Principle Part 1

Steve Smith
http://pluralsight.com/

Outline

- DIP Defined
- The Problem
- An Example
- Refactoring to Apply DIP
- Related Fundamentals

DIP: The Dependency Inversion Principle

High-level modules should not depend on low-level modules. Both should depend on abstractions.

Abstractions should not depend on details. Details should depend on abstractions.

Agile Principles, Patterns, and Practices in C#

DEPENDENCY INVERSION PRINCIPLE

Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

What are dependencies?

- Framework
- Third Party Libraries
- Database
- File System
- Email
- Web Services
- System Resources (Clock)
- Configuration
- The new Keyword
- Static methods
- Thread.Sleep
- Random

Traditional Programming and Dependencies

- **High Level Modules Call Low Level Modules**

- **User Interface depends on**
  - Business Logic depends on
    - Infrastructure
    - Utility
    - Data Access

- **Static methods are used for convenience or as Façade layers**
- **Class instantiation / Call stack logic is scattered through all modules**
  - Violation of Single Responsibility Principle

Class Dependencies: Be Honest!

- Class constructors should require any dependencies the class needs
- Classes whose constructors make this clear have *explicit* dependencies
- Classes that do not have *implicit*, hidden dependencies

```
public class HelloWorldHidden
{
    public string Hello(string name)
    {
        if (DateTime.Now.Hour < 12) return "Good morning, " + name;
        if (DateTime.Now.Hour < 18) return "Good afternoon, " + name;
        return "Good evening, " + name;
    }
}
```

Classes Should Declare What They Need

```
public class HelloWorldExplicit
{
    private readonly DateTime _timeOfGreeting;

    public HelloWorldExplicit(DateTime timeOfGreeting)
    {
        _timeOfGreeting = timeOfGreeting;
    }

    public string Hello(string name)
    {
        if (_timeOfGreeting.Hour < 12) return "Good morning, " + name;
        if (_timeOfGreeting.Hour < 18) return "Good afternoon, " + name;
        return "Good evening, " + name;
    }
}
```

Demo

Violating DIP

The Problem

- **Order has hidden dependencies:**
  - MailMessage
  - SmtpClient
  - InventorySystem
  - PaymentGateway
  - Logger
  - DateTime.Now

- **Result**
  - Tight coupling
  - No way to change implementation details (OCP violation)
  - Difficult to test

Dependency Injection

- ***Dependency Injection*** **is a technique that is used to allow calling code to *inject* the dependencies a class needs when it is instantiated.**

- **The Hollywood Principle**
  - □ "Don't call us; we'll call you"

- **Three Primary Techniques**
  - □ Constructor Injection
  - □ Property Injection
  - □ Parameter Injection

- **Other methods exist as well**

Constructor Injection

Strategy Pattern

- **Dependencies are passed in via constructor**

- **Pros**
  - Classes self-document what they need to perform their work
  - Works well with or without a container
  - Classes are always in a valid state once constructed

- **Cons**
  - Constructors can have many parameters/dependencies (design smell)
  - Some features (e.g. Serialization) may require a *default constructor*
  - Some methods in the class may not require things other methods require (design smell)

Property Injection

- **Dependencies are passed in via a property**
  - Also known as “setter injection”

- **Pros**
  - Dependency can be changed at any time during object lifetime
  - Very flexible

- **Cons**
  - Objects may be in an invalid state between construction and setting of dependencies via setters
  - Less intuitive

Parameter Injection

- **Dependencies are passed in via a method parameter**

- **Pros**
  - Most granular
  - Very flexible
  - Requires no change to rest of class

- **Cons**
  - Breaks method signature
  - Can result in many parameters (design smell)

- ***Consider if only one method has the dependency, otherwise prefer constructor injection***

Refactoring

- Extract Dependencies into Interfaces
- Inject implementations of interfaces into Order
- Reduce Order’s responsibilities (apply SRP)

Demo

Refactoring to a Better Design

DIP Smells

- Use of new keyword

```
foreach(var item in cart.Items)
{
  try
  {
    var inventorySystem = new InventorySystem();
    inventorySystem.Reserve(item.Sku, item.Quantity);
  }
}
```

DIP Smells

- Use of static methods/properties

```
message.Subject = "Your order placed on " +
   DateTime.Now.ToString();
```

Or

```
DataAccess.SaveCustomer(myCustomer);
```

Where do we instantiate objects?

- **Applying Dependency Injection typically results in many interfaces that eventually need to be instantiated *somewhere… but where?***

- **Default Constructor**
  - □ You can provide a default constructor that news up the instances you expect to typically need in your application
  - □ Referred to as "poor man's dependency injection" or "poor man's IoC"

- **Main**
  - □ You can manually instantiate whatever is needed in your application's startup routine or main() method

- **IoC Container**
  - □ Use an "Inversion of Control" Container

IoC Containers

- **Responsible for object graph instantiation**
- **Initiated at application startup via code or configuration**
- **Managed interfaces and the implementation to be used are *Registered* with the container**
- **Dependencies on interfaces are *Resolved* at application startup or runtime**

- **Examples of IoC Containers for .NET**
  - Microsoft Unity
  - StructureMap
  - Ninject
  - Windsor
  - Funq / Munq

Summary

- **Depend on abstractions.**
- **Don’t force high-level modules to depend on low-level modules through direct instantiation or static method calls**
- **Declare class dependencies explicitly in their constructors**
- **Inject dependencies via constructor, property, or parameter injection**
- **Related Fundamentals:**
  - Single Responsibility Principle
  - Interface Segregation Principle
  - Façade Pattern
  - Inversion of Control Containers
- **Recommended Reading:**
  - Agile Principles, Patterns, and Practices by Robert C. Martin and Micah Martin [http://amzn.to/agilepppcsharp]
  - http://www.martinfowler.com/articles/injection.html

Credits

- **Images Used Under License**
  - http://www.lostechies.com/blogs/derickbailey/archive/2009/02/11/solid-development-principles-in-motivational-pictures.aspx

For more in-depth online developer training visit

pluralsight
see what you can learn

on-demand content from authors you trust