SOLID

Software Development is not a Jenga game

S.O.L.I.D. Software Development

Achieving Object Oriented Principles, One Step At A Time

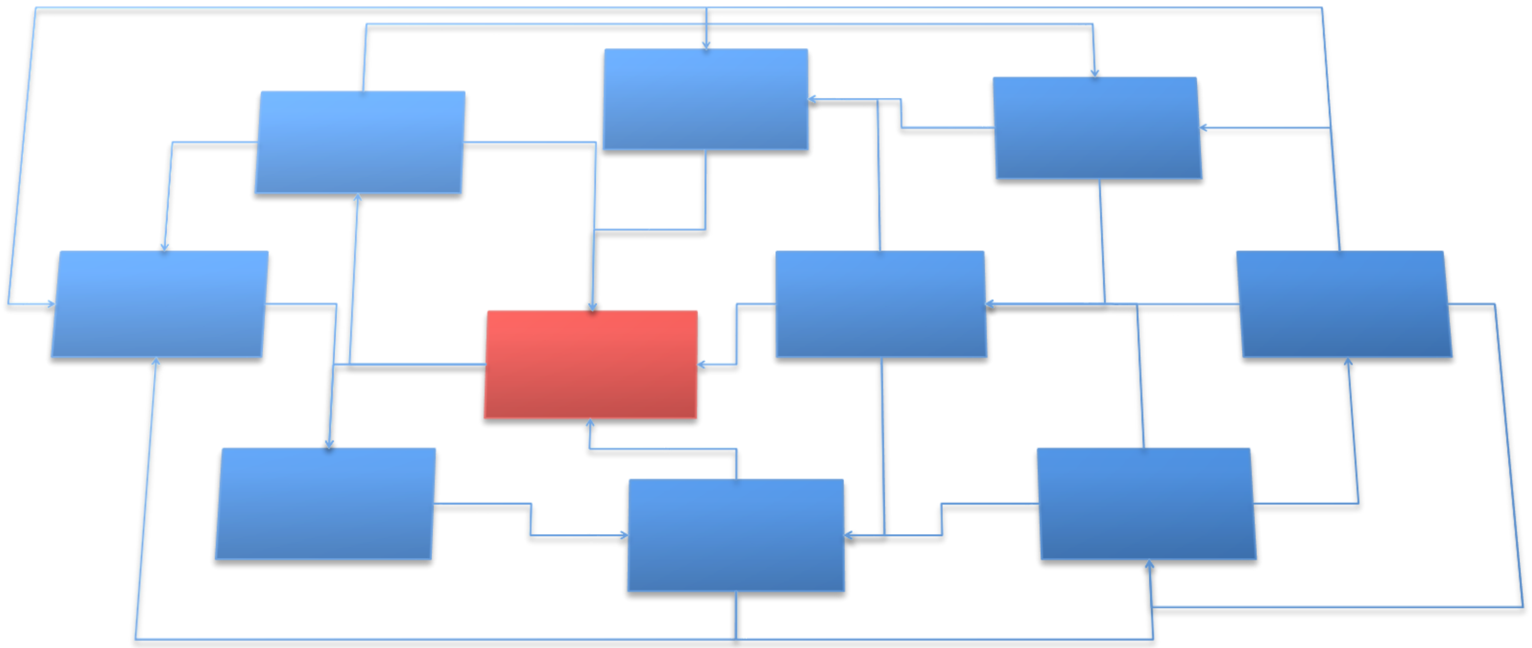
Object Oriented Principles

Cohesion:

"A measure of how strongly-related and focused the various responsibilities of a software module are" - Wikipedia

Object Oriented Principles

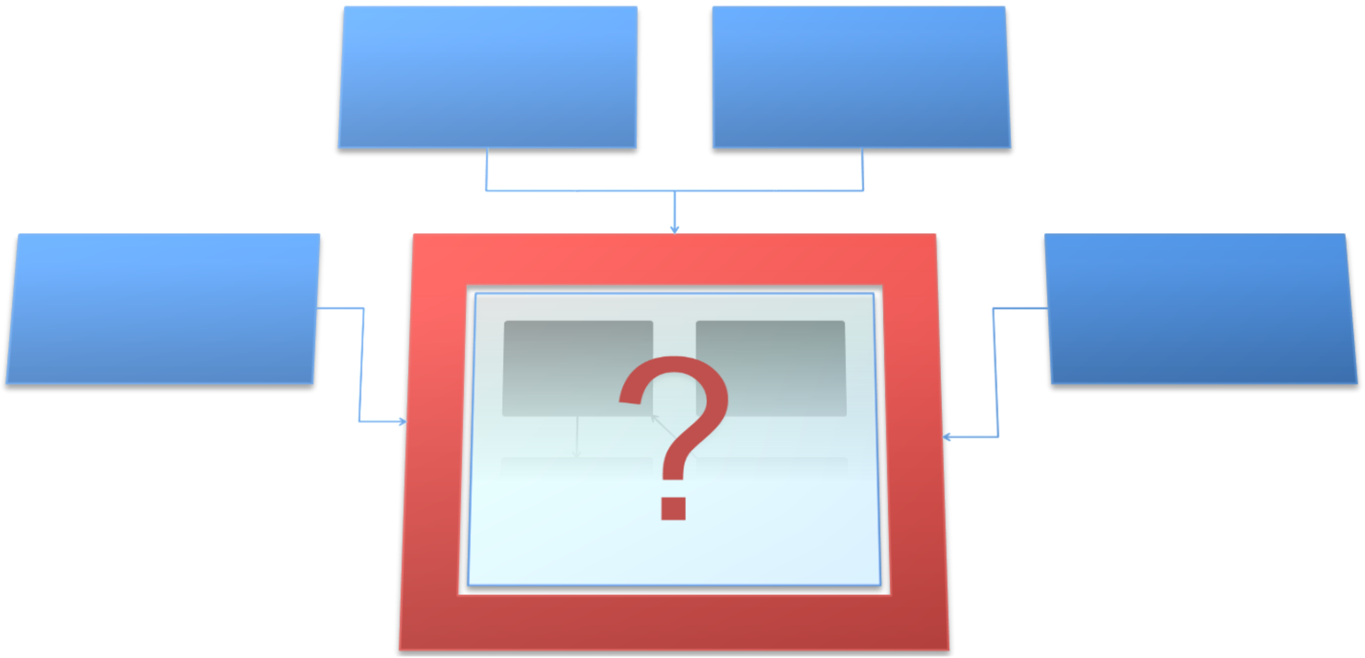
- Coupling:
  - "The degree to which each program module relies on each one of the other modules" – Wikipedia



Object Oriented Principles

Encapsulation:

“The hiding of *design decisions* in a computer program that are most likely to change” - Wikipedia



S.O.L.I.D. Principles

SRP: Single Responsibility Principle

OCP: Open Closed Principle

LSP: Liskov Substitution Principle

ISP: Interface Segregation Principle

DIP: Dependency Inversion Principle

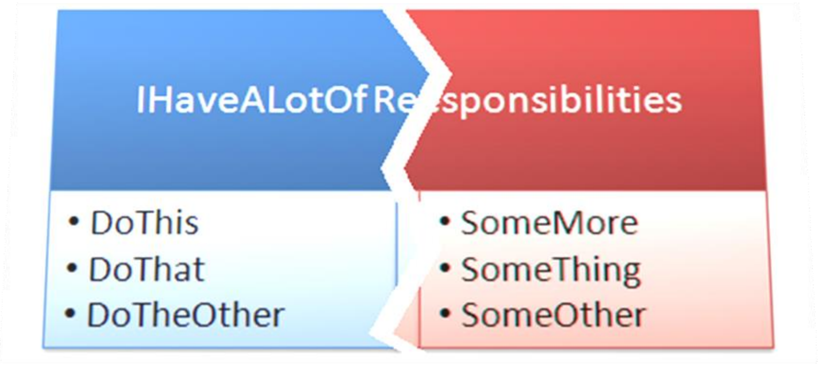
SRP: Single Responsibility Principle

One Responsibility – One Reason To Change

SRP: Single Responsibility

"If a class has more then one responsibility, then the responsibilities become coupled. Changes to one responsibility may impair or inhibit the class' ability to meet the others. This kind of coupling leads to fragile designs that break in unexpected ways when changed."

- Robert C. Martin

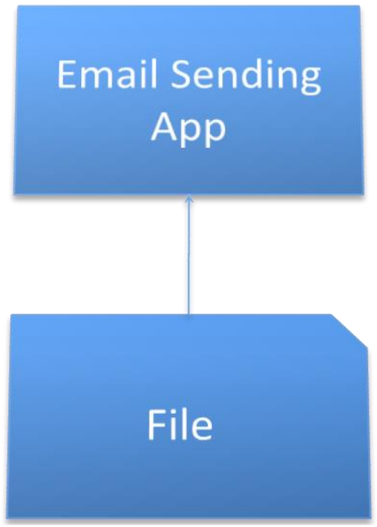


SINGLE RESPONSIBILITY PRINCIPLE

Just Because You Can, Doesn't Mean You Should

SRP: Single Responsibility

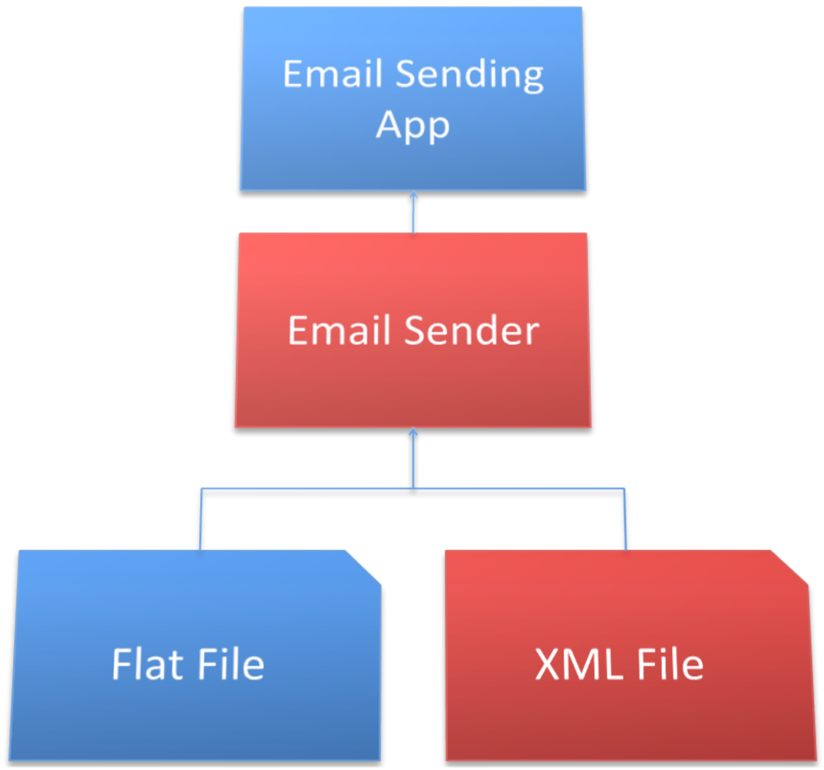
Example App: Read A Flat File And Send An Email



SRP: Single Responsibility

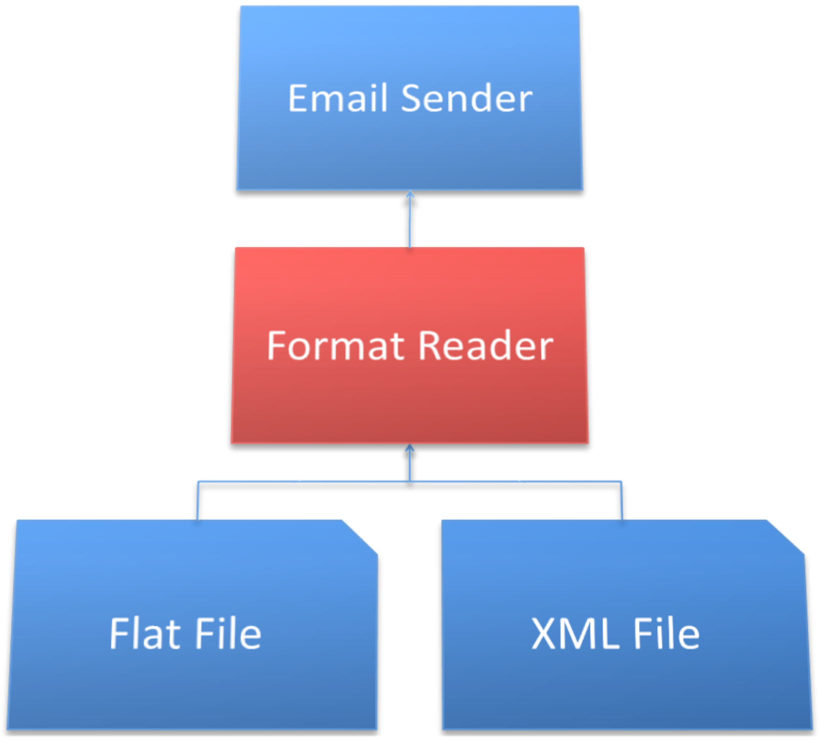
Example App New Requirements:

- Send From Non-WinForms App.
- Read XML Or Flat File



SRP: Single Responsibility

Example App: A Better Structure



OCP: Open Closed Principle

Open For Extension, Closed For Modification

OCP: Open Closed Principle

Modules that conform to the open-closed principle have two primary attributes.

1. *They are "Open For Extension".*
   This means that the behavior of the module can be extended. That we can make the module behave in new and different ways as the requirements of the application change, or to meet the needs of new applications.

2. *They are "Closed for Modification".*
   The source code of such a module is inviolate. No one is allowed to make source code changes to it.

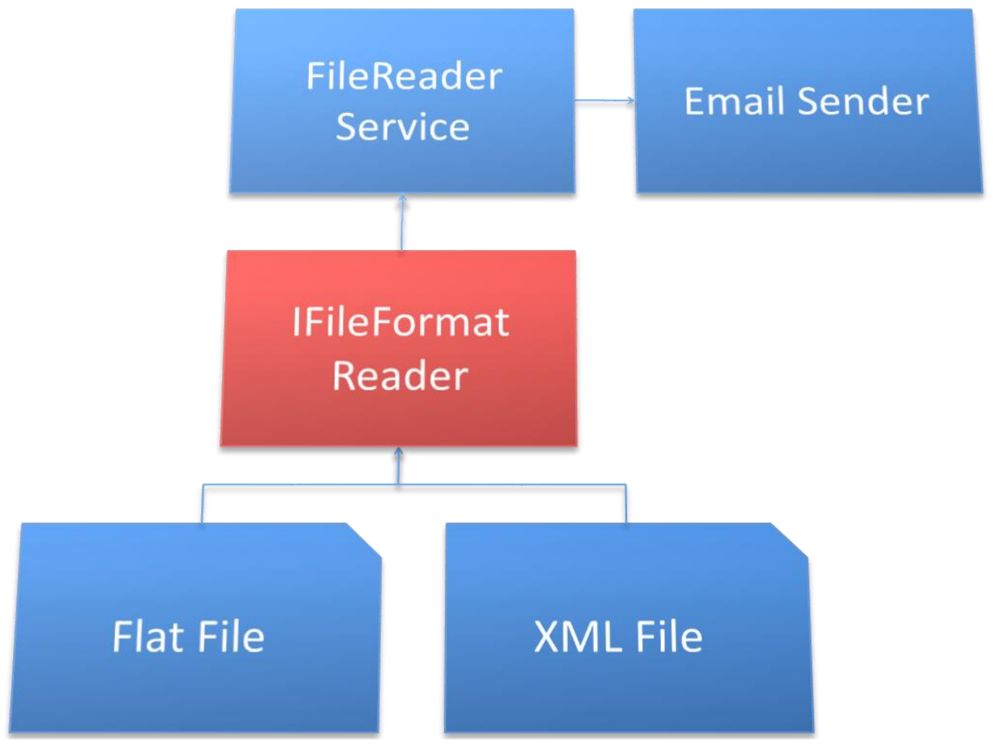
- Robert C. Martin

OPEN CLOSED PRINCIPLE

Open Chest Surgery Is Not Needed When Putting On A Coat

OCP: Open Closed Principle

Example App: Restructuring For OCP



LSP: Liskov Substitution Principle

Derived Classes Must Be Semantically Substitutable For Their Base Classes.

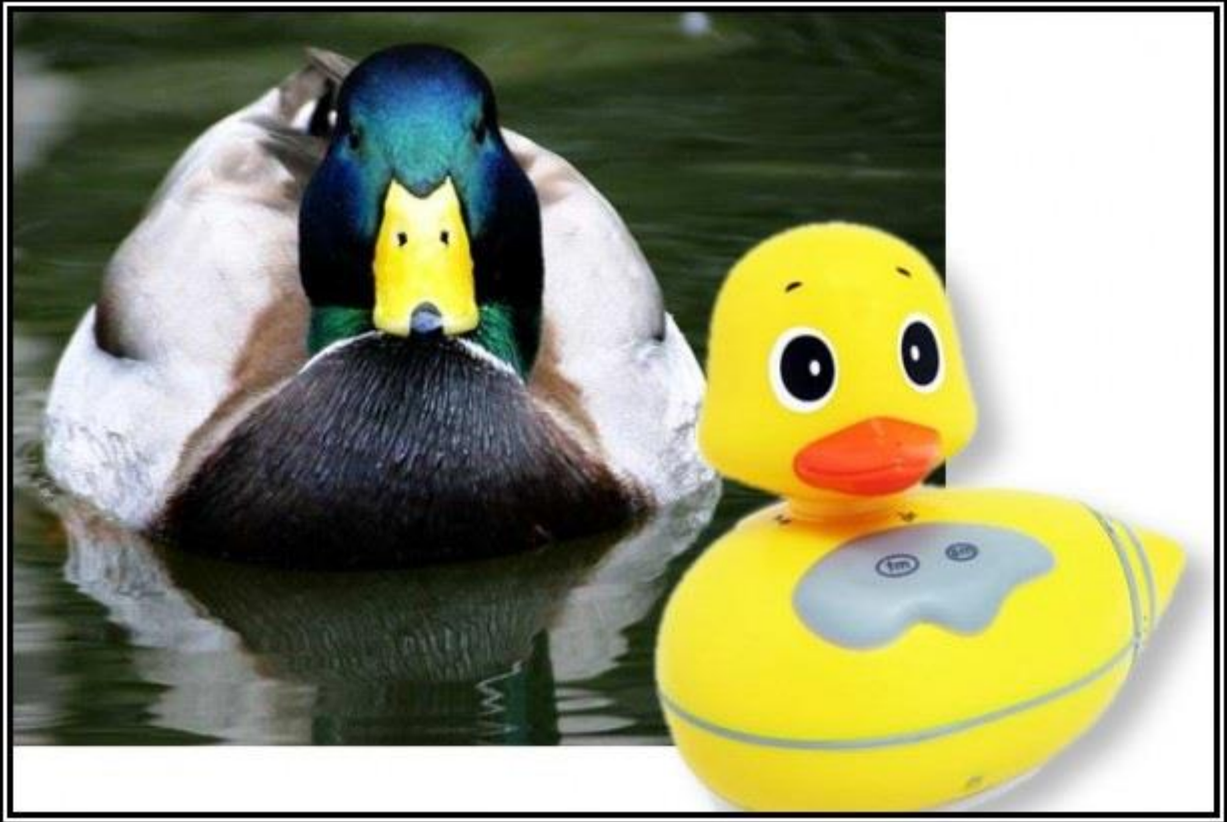
LSP: Liskov Substitution Principle

"If for each object o1 of type S there is an object o2 of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when o1 is substituted for o2 then S is a subtype of T."

- Barbara Liskov

Square != Rectangle

!=

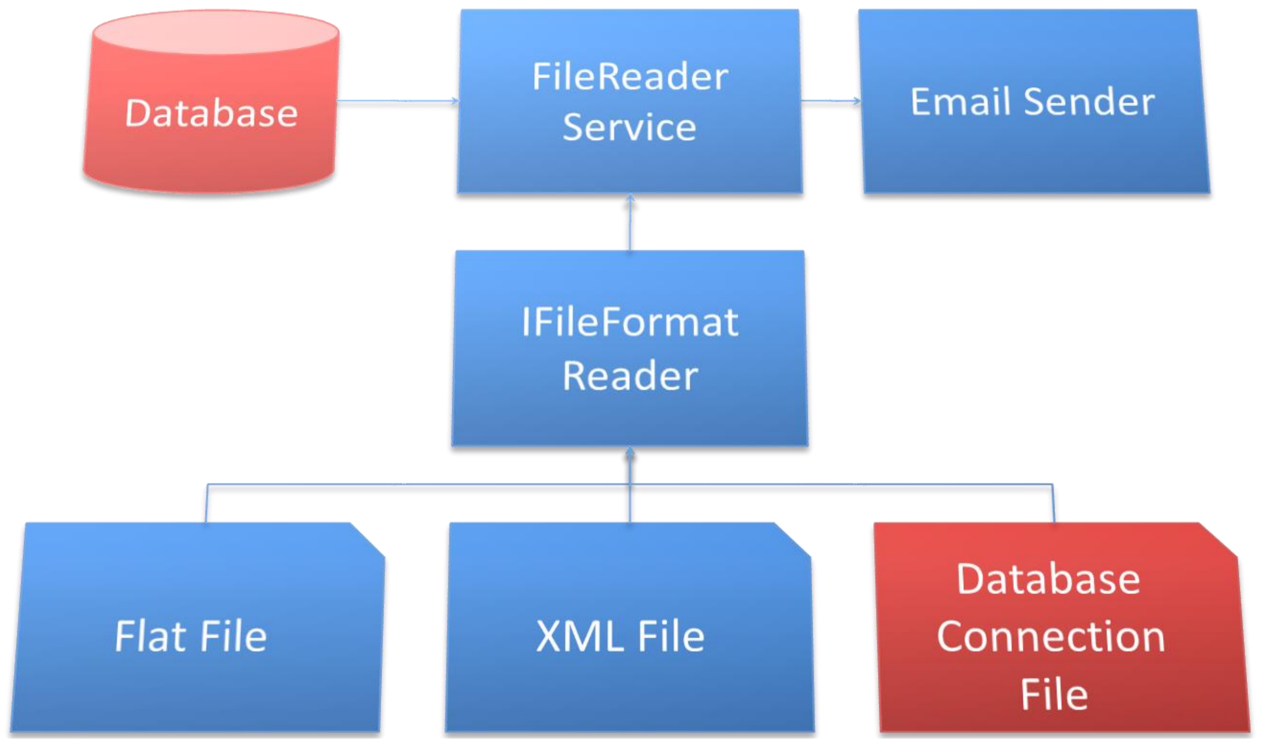


LISKOV SUBSTITUTION PRINCIPLE

If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

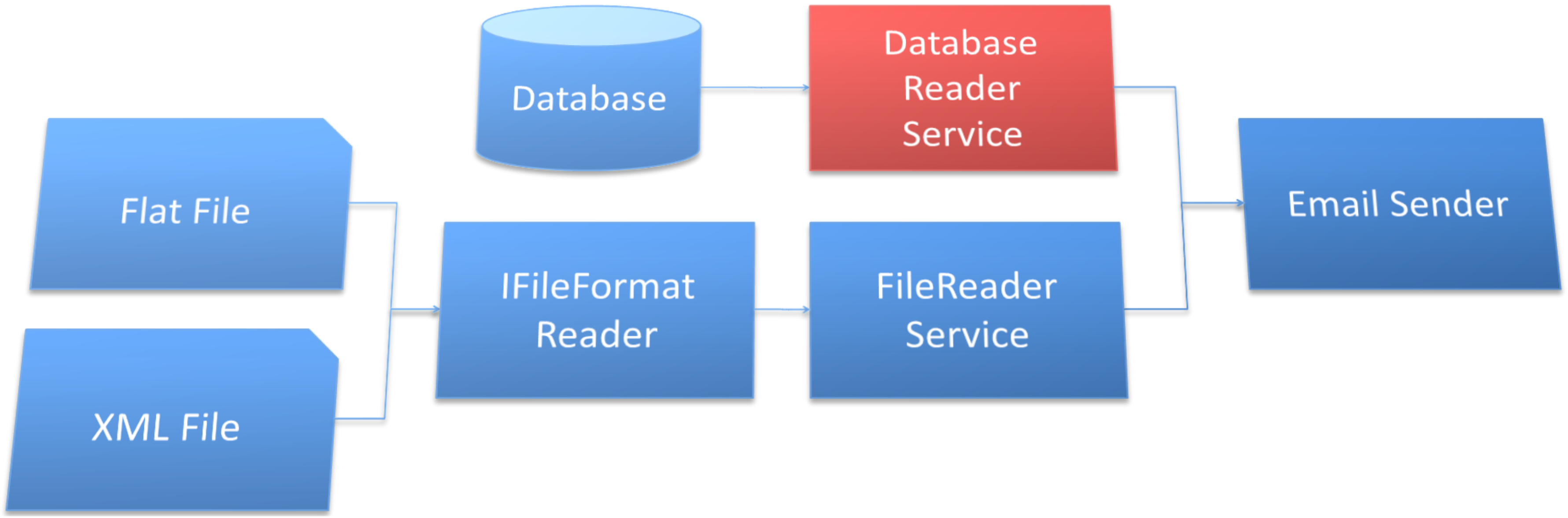
LSP: Liskov Substitution Principle

Example App: Violating LSP with Database Connection Info



LSP: Liskov Substitution Principle

Example App: Correcting For LSP – Move The Database Reader



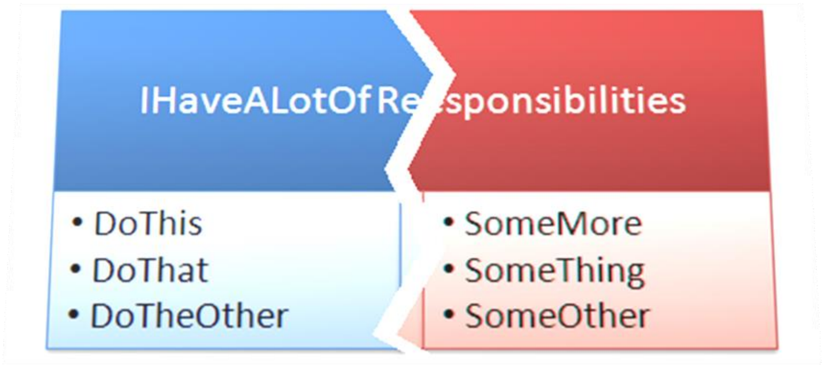
ISP: Interface Segregation Principle

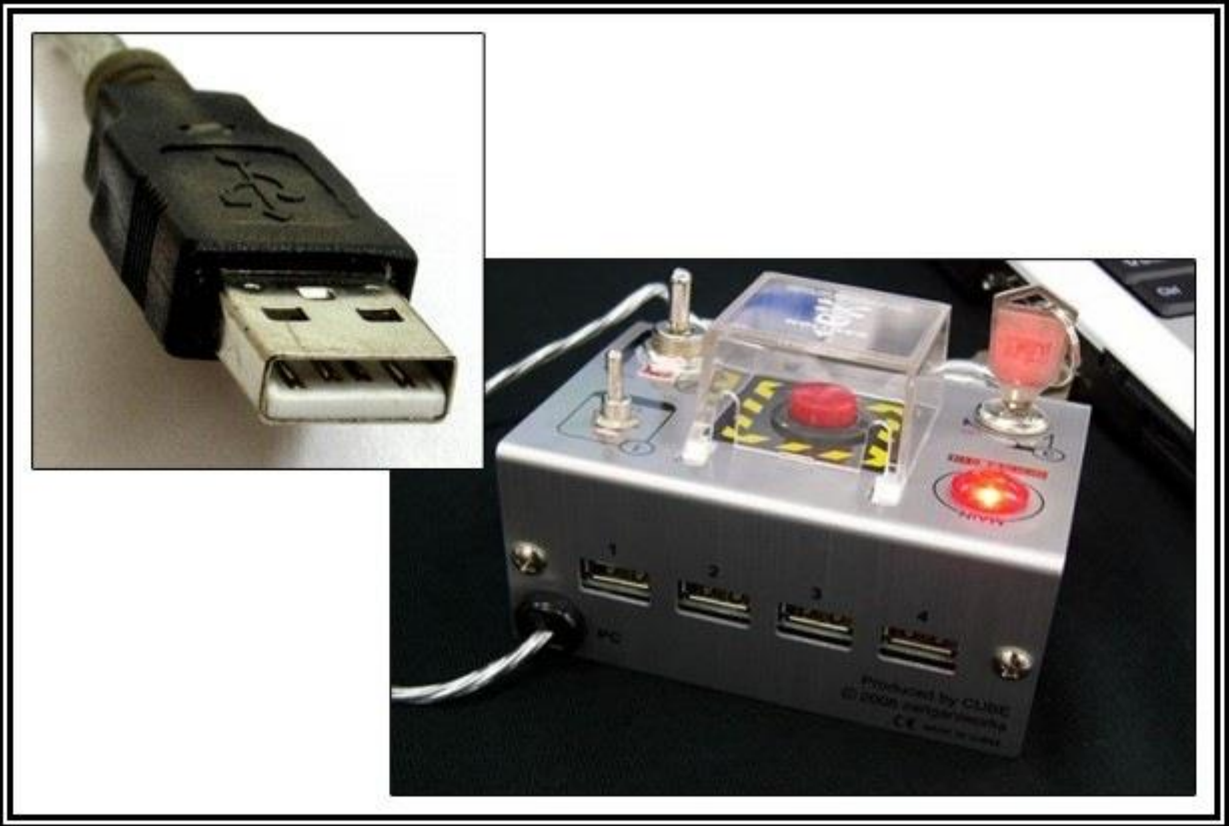
A Client Should Not Depend On An Interface It Does Not Use

ISP: Interface Segregation Principle

“This principle deals with the disadvantages of ‘fat’ interfaces. Classes that have ‘fat’ interfaces are classes whose interfaces are not cohesive. In other words, the interfaces of the class can be broken up into groups of member functions. Each group serves a different set of clients. Thus some clients use one group of member functions, and other clients use the other groups.”

- Robert Martin



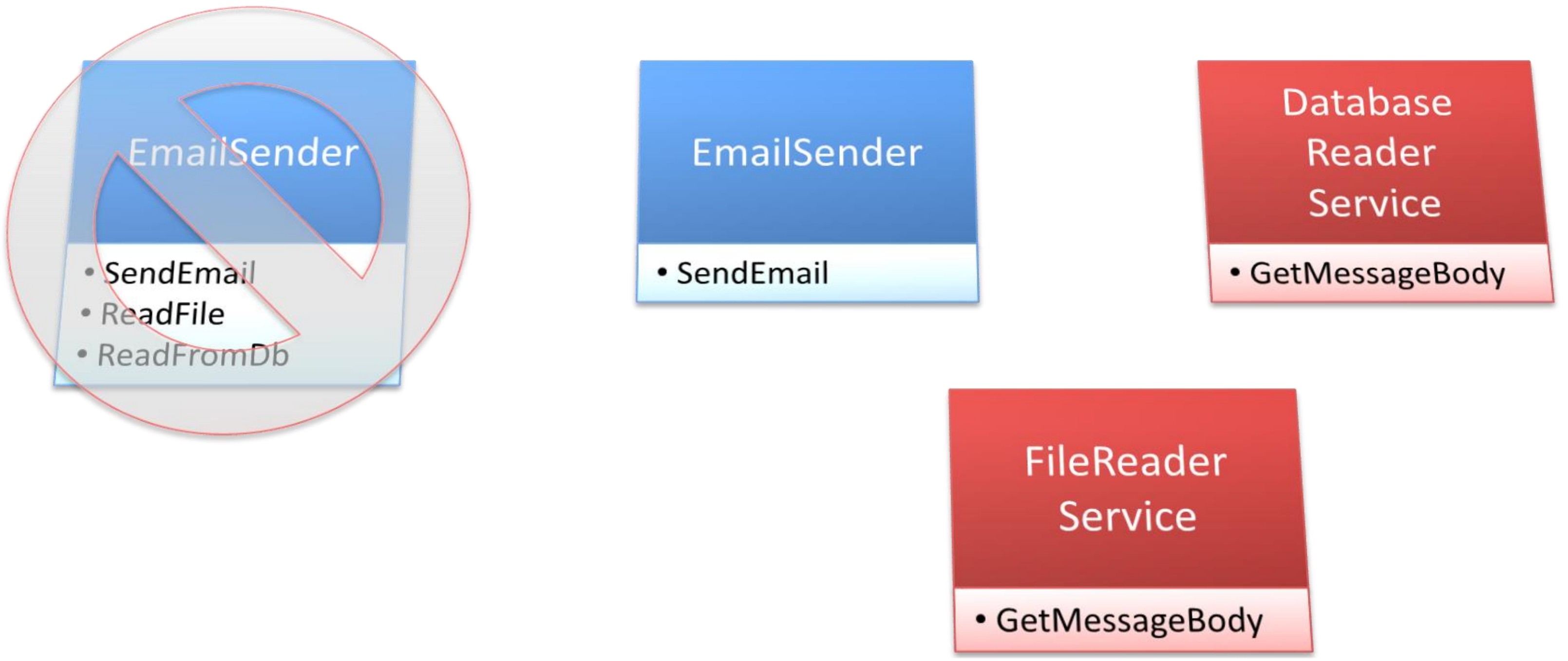


INTERFACE SEGREGATION PRINCIPLE

You Want Me To Plug This In, Where?

ISP: Interface Segregation Principle

Example App: Clarifying The Email Sender and Message Info Parsing



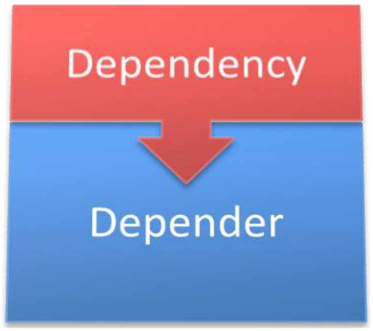
DIP: Dependency Inversion Principle

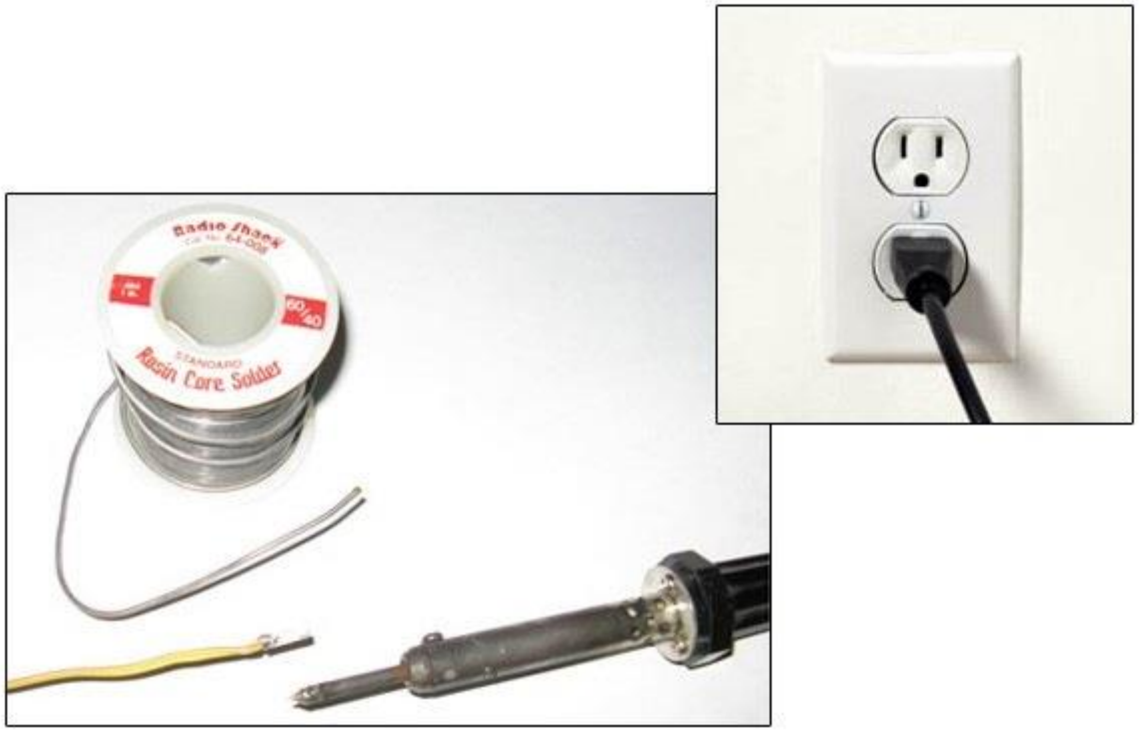
Depend On Abstractions, Not Concrete Details And Implementations

DIP: Dependency Inversion Principle

“What is it that makes a design rigid, fragile and immobile? It is the interdependence of the modules within that design. A design is rigid if it cannot be easily changed. Such rigidity is due to the fact that a single change to heavily interdependent software begins a cascade of changes in dependent modules.”

- Robert Martin

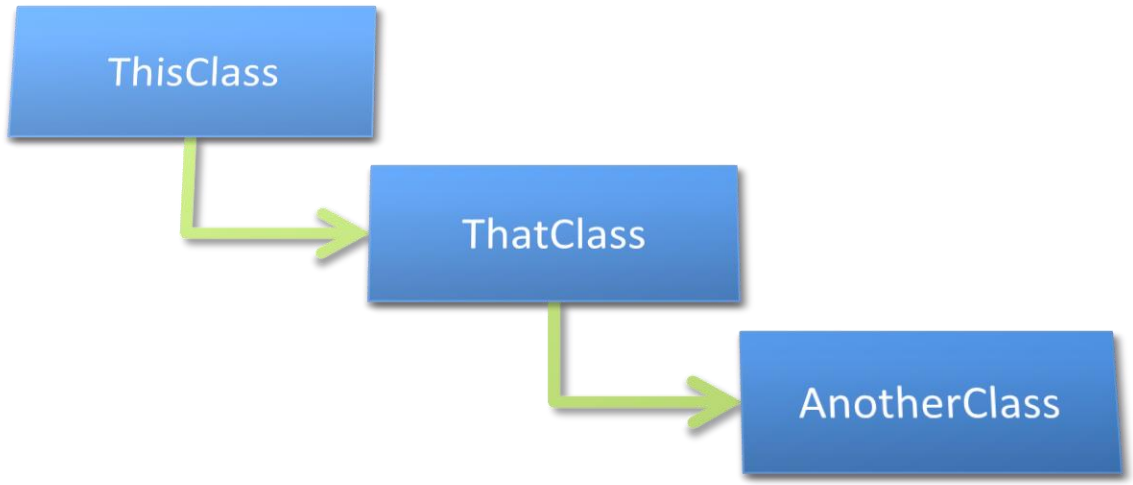




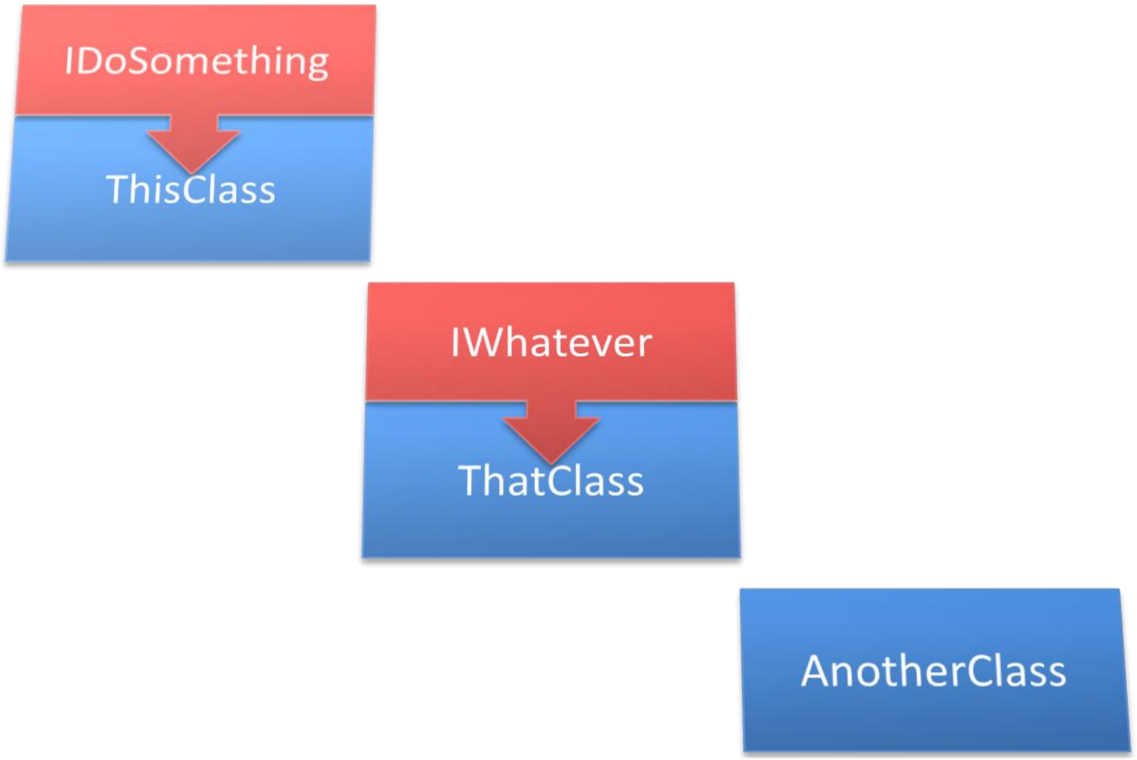
DEPENDENCY INVERSION PRINCIPLE

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

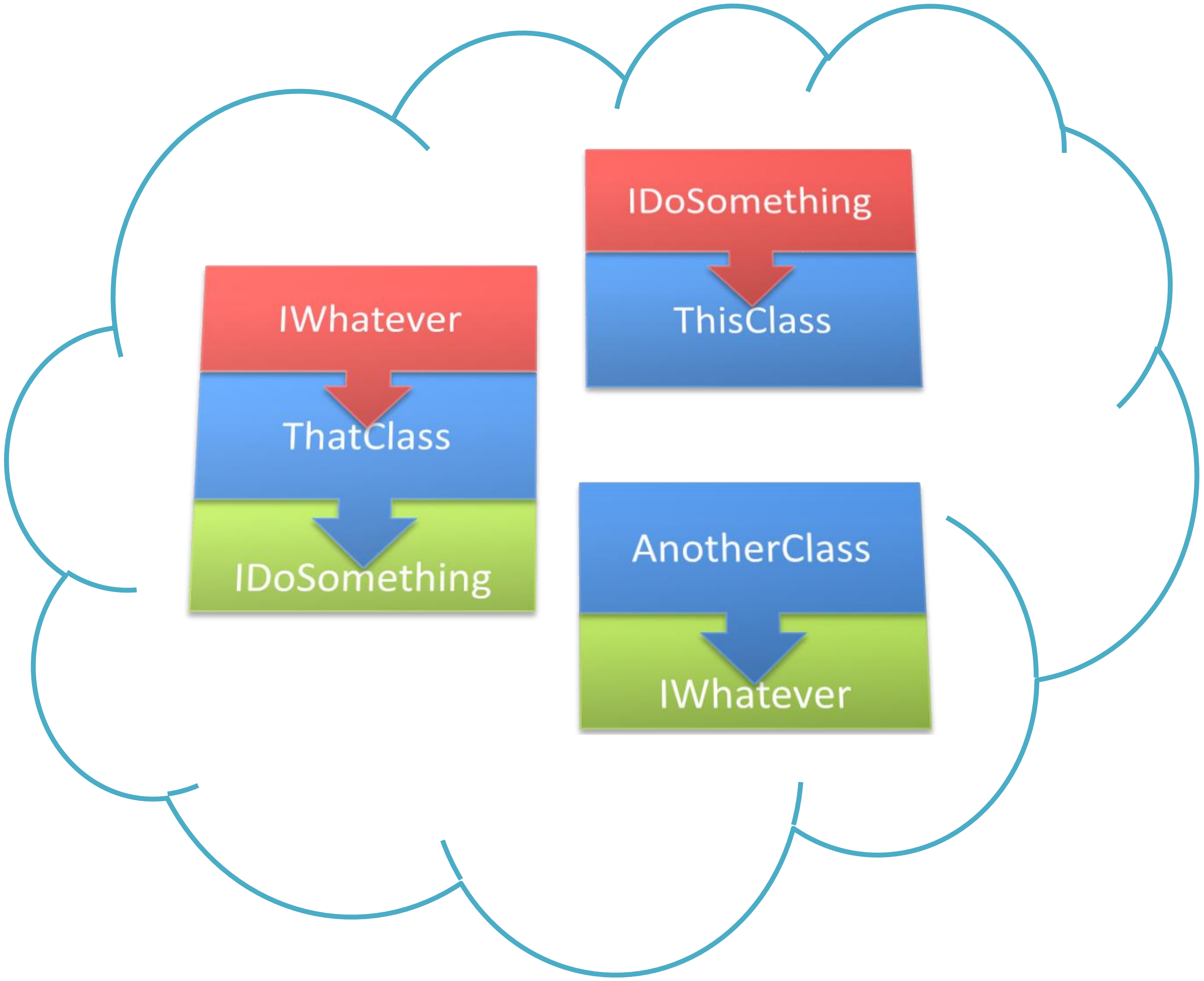
DIP: Dependency Inversion Principle



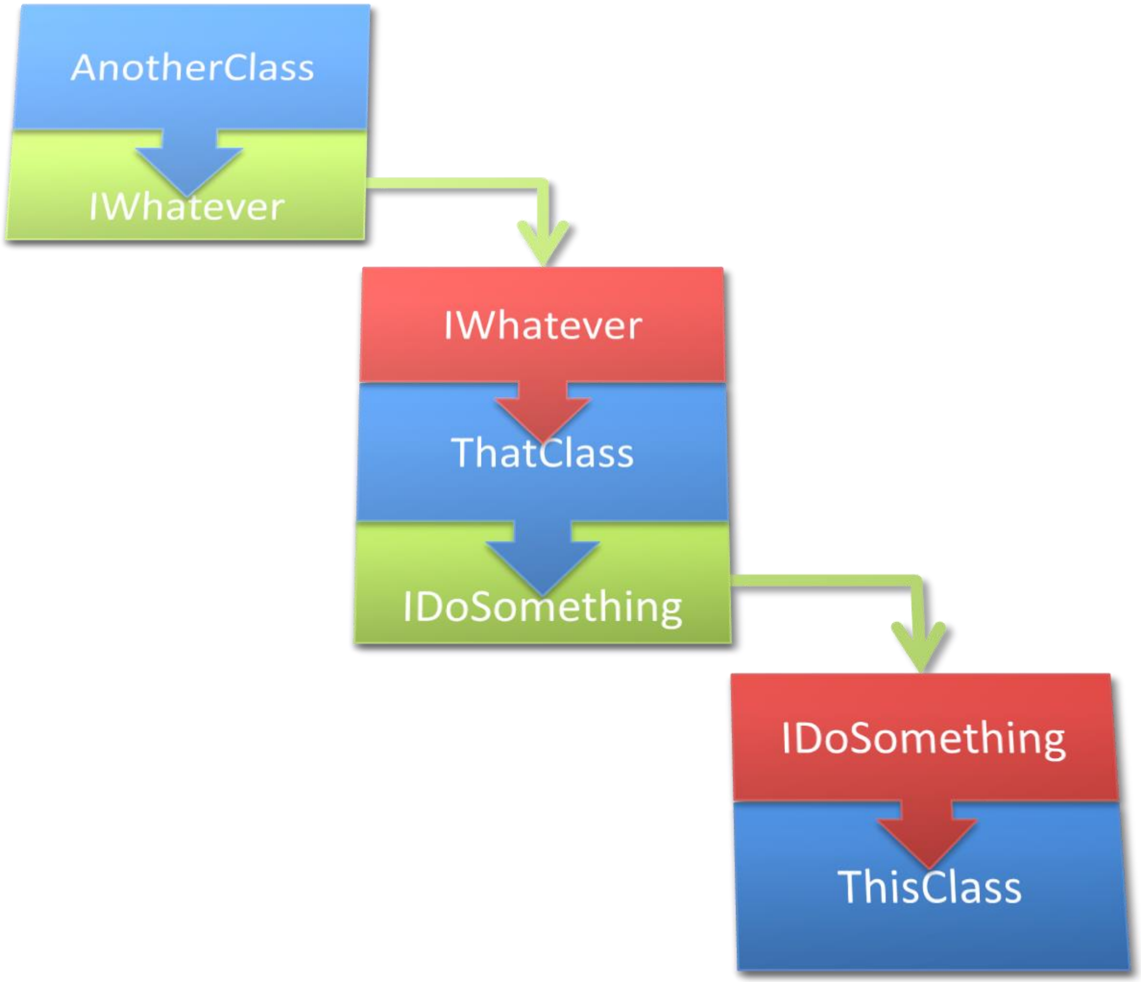
DIP: Dependency Inversion Principle



DIP: Dependency Inversion Principle

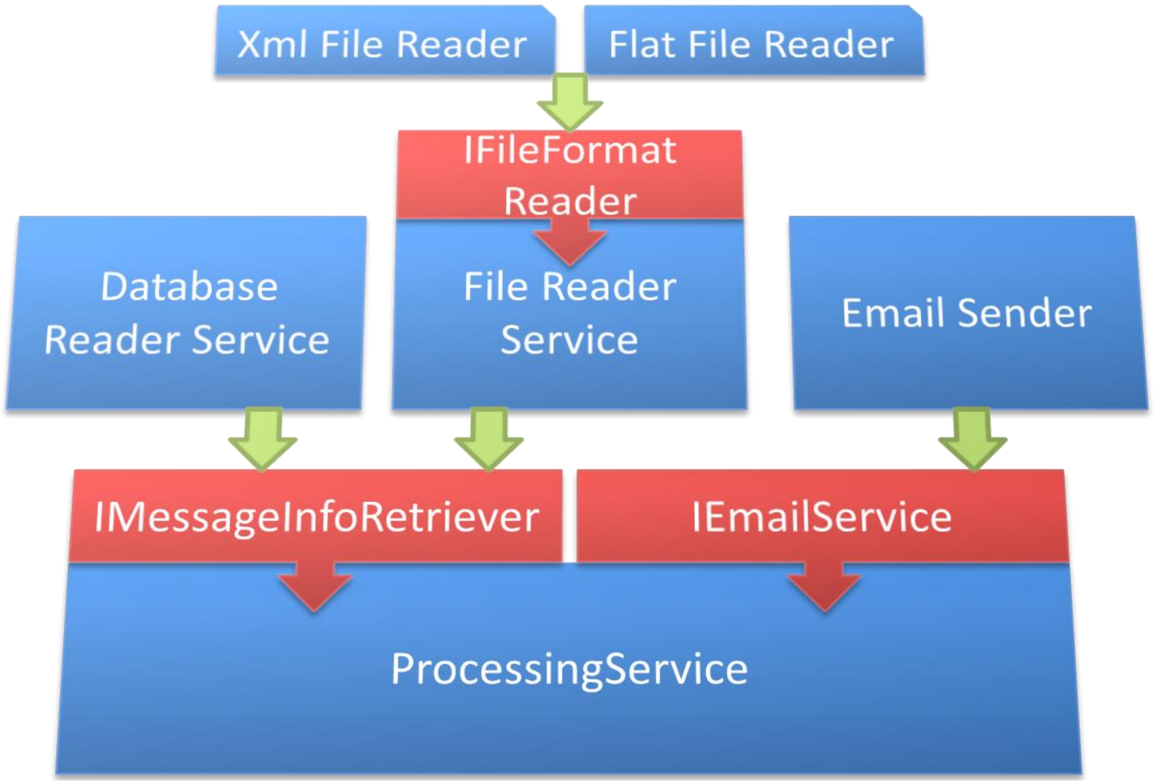


DIP: Dependency Inversion Principle



DIP: Dependency Inversion Principle

Example App: Constructor Dependencies in a Processing Service

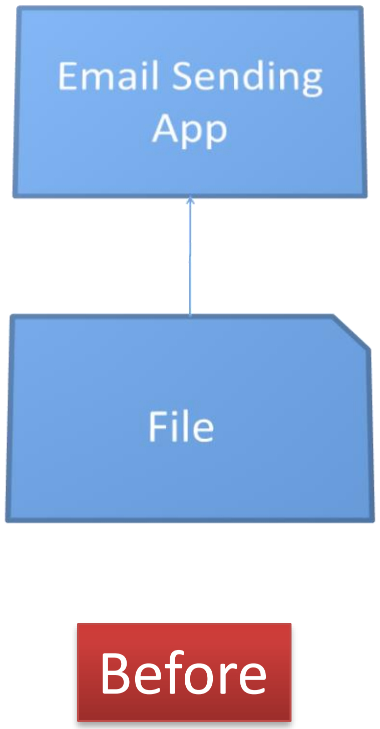


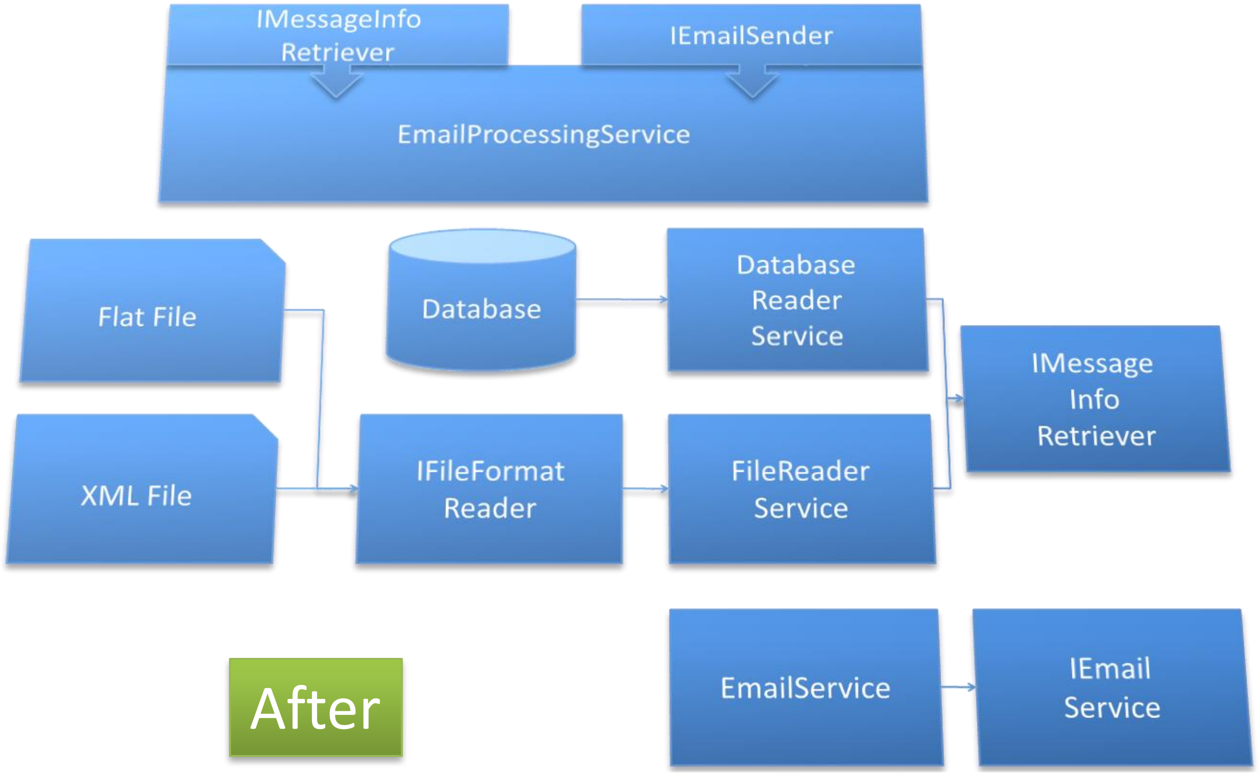
Summarizing Our S.O.L.I.D. Conversion

Compare And Contrast The Original Code To The New Code

S.O.L.I.D. Conversion Summary

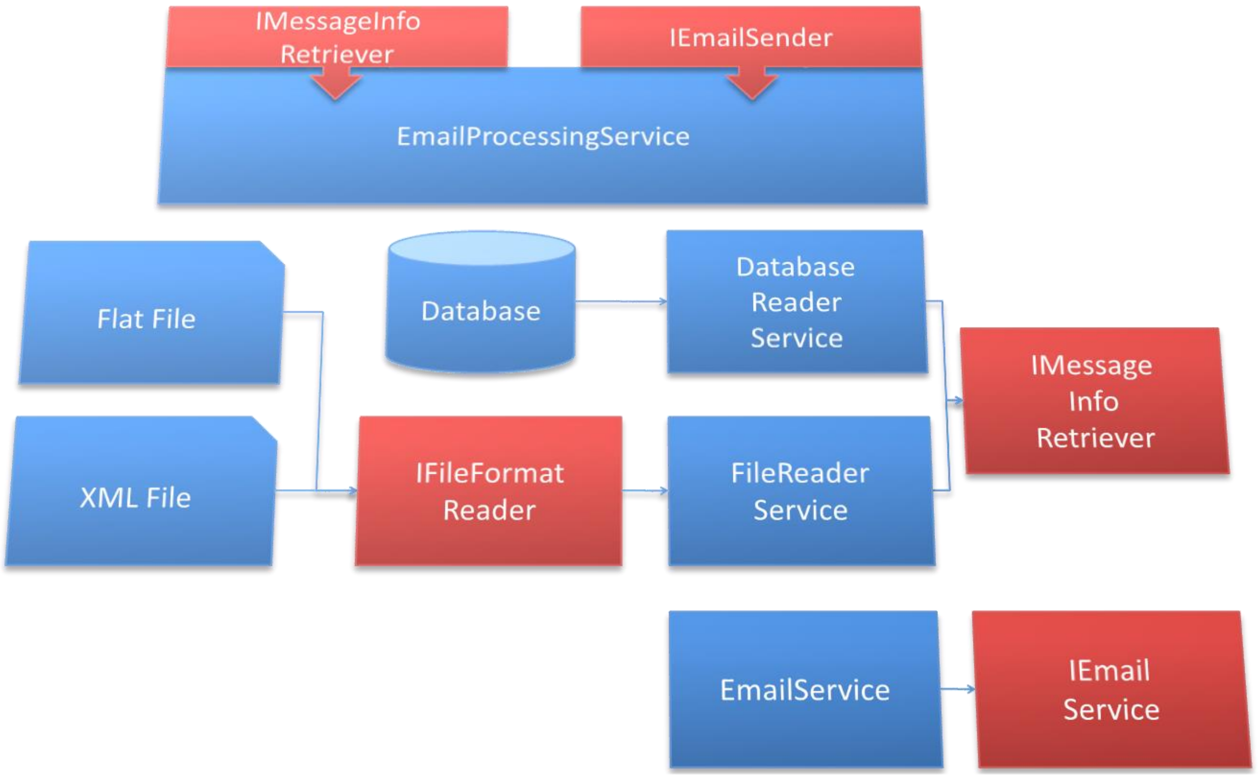
Example App: Before And After





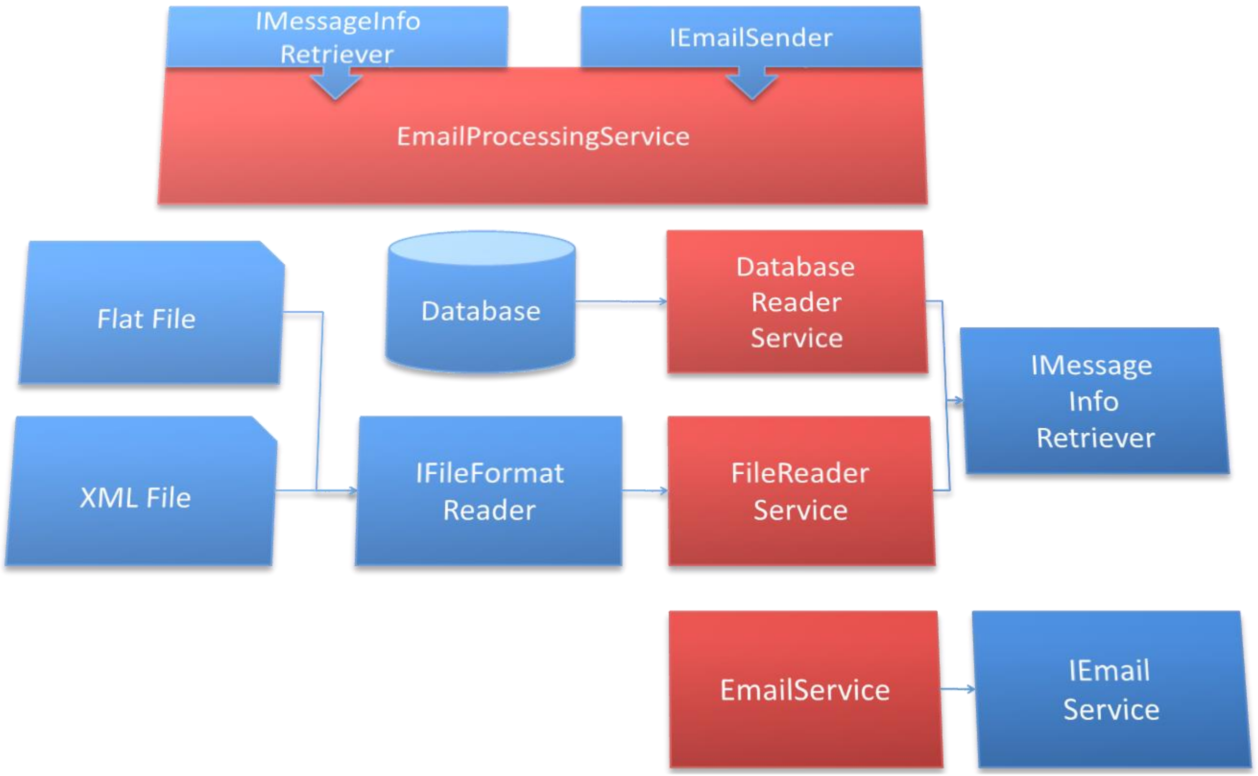
S.O.L.I.D. -> OO Principles

Low Coupling: OCP, DIP, ISP



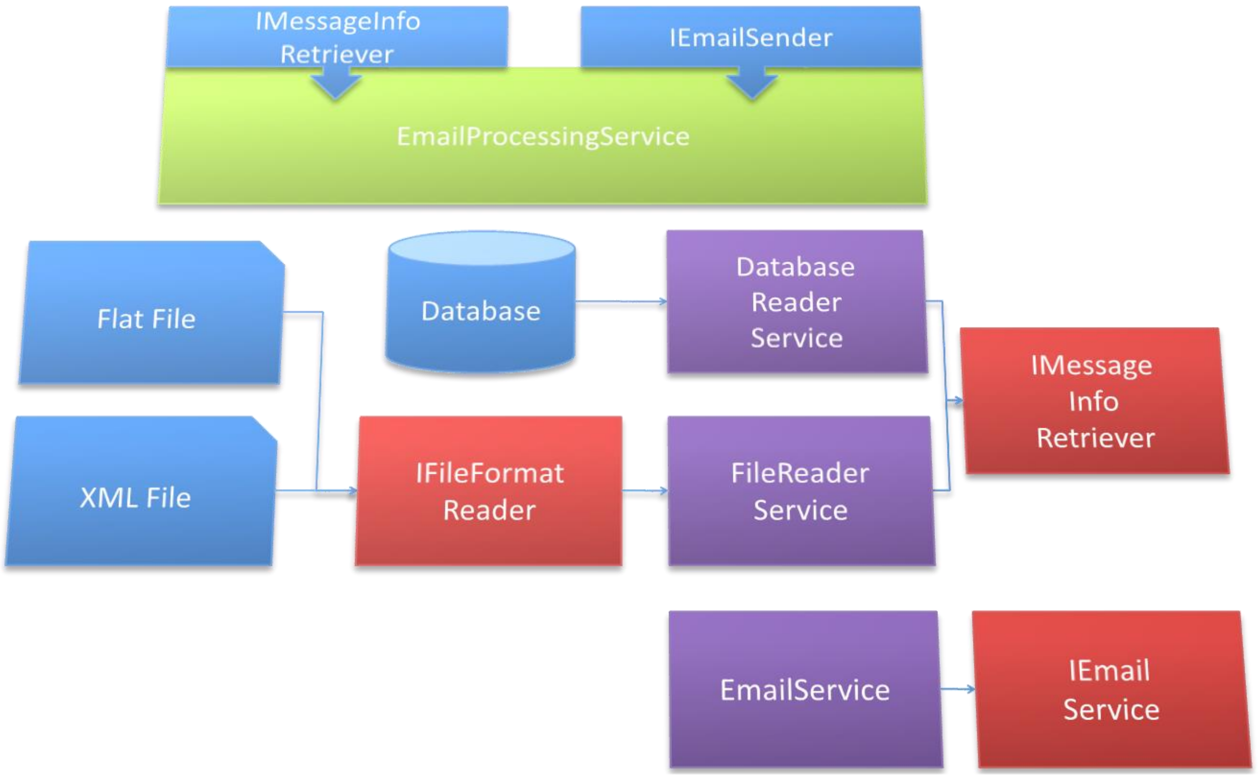
S.O.L.I.D. -> OO Principles

High Cohesion: Low Coupling + SRP, LSP



S.O.L.I.D. -> OO Principles

Encapsulation: SRP, LSP, DIP



Additional Resources

Uncle Bob's Principle Of Object Oriented Development:

http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod

Pablo's Topic Of The Month: SOLID

http://www.lostechies.com/blogs/chad_myers/archive/2008/03/07/pablo-s-topic-of-the-month-march-solid-principles.aspx

Agile Principles, Patterns, And Practices In C#

by Robert (Uncle Bob) Martin and Micah Martin