S.O.L.I.D Design Principles

Venkat Shiva Reddy

SOLID Seems to be Everywhere These Days

What is SOLID?

- **SOLID**
  - **Single Responsibility**
  - **Open-Closed**
  - **Liskov Substitution**
  - **Interface Segregation**
  - **Dependency Inversion**

....is a mnemonic acronym introduced by **Michael Feathers** for the "first five principles" identified by **Robert C. Martin** in the early 2000, that stands for five basic principles of object-oriented programming and design.

SRP — Single Responsibility Principle

OCP — Open Closed Principle

LSP — Liskov Substitution Principle

ISP — Interface Segregation Principle

DIP — Dependency Inversion Principle

SRP	There *should never be more than one reason for a class to change*
OCP	*Software entities should be open for extension, but closed for modification.*
LSP	*Functions that use references to base classes must be able to use objects of derived classes without knowing it.*
ISP	*Clients should not be forced to depend upon interfaces that they do not use.*
DIP	*High level modules should not depend upon low level modules. Both should depend upon abstractions.*

Single Responsibility Principle
Just because you *can* doesn't mean you *should.*

SRP – Single Responsibility

- **The "S" in SOLID is for *Single Responsibility Principle,*** which states that every object should have a single responsibility and that all of its services should be aligned with that responsibility.

- *There should never be more than one reason for a class to change.*

- *A class should concentrate on doing one thing*

SRP

- For example, an Invoice class might have the responsibility of calculating various amounts based on it’s data. In that case it probably shouldn’t know about how to retrieve this data from a database, or how to format an invoice for print or display.

- It would be a bad design to couple two things that change for different reasons at different times.

- Violations of the SRP are pretty easy to notice: the class seems to be doing too much, is too big and too complicated. The easiest way to fix this is to split the class.

RSP

Open-Closed Principle
Open-chest surgery isn't needed when putting on a coat.

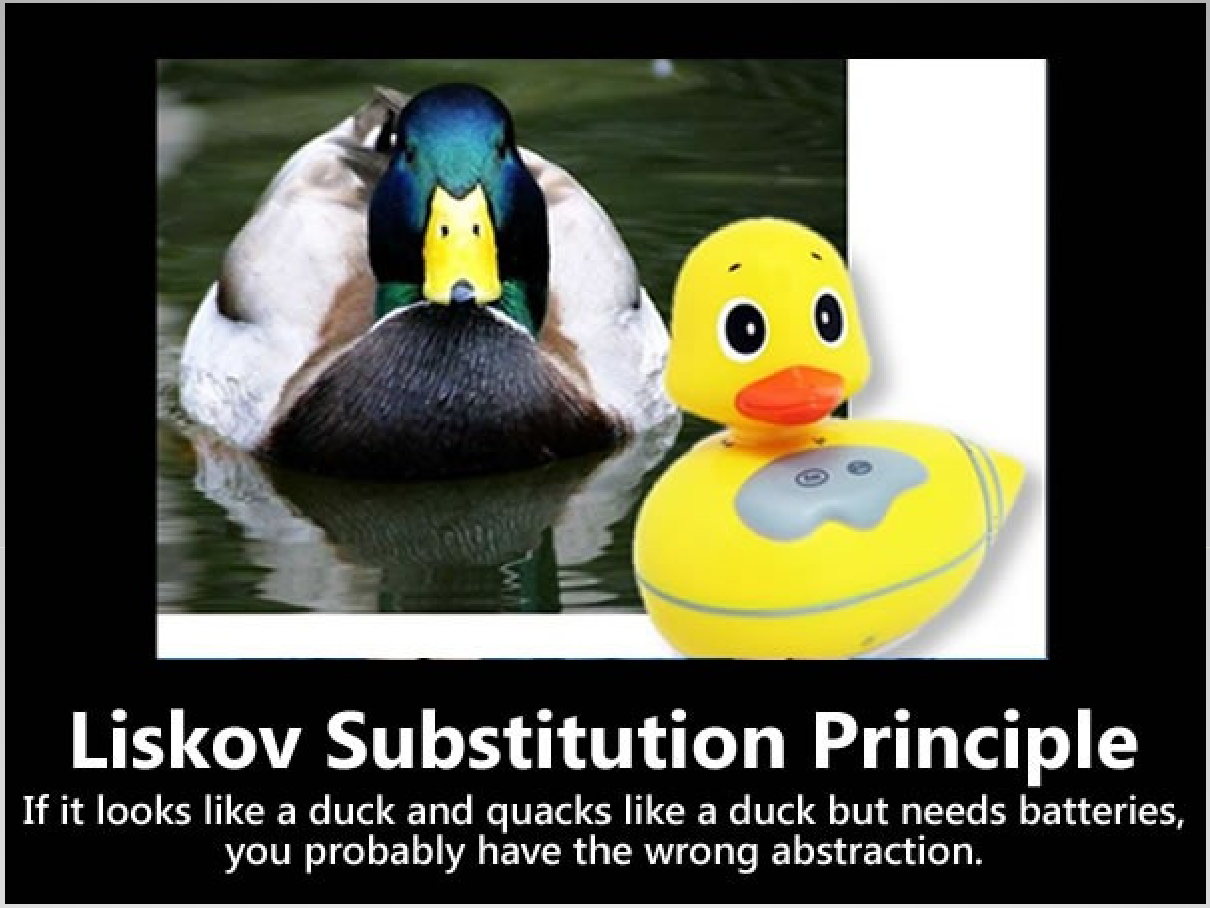
OCP

- **The "O" in SOLID is for *Open-Closed Principle*,** which states that software entities – such as classes, modules, functions and so on – should be open for extension but closed for modification.

- The idea is that it's often better to make changes to things like classes by adding to or building on top of them (using mechanisms like sub-classing or polymorphism) rather than modifying their code.

- *Change a class' behaviour using inheritance and composition*

OCP


Liskov Substitution Principle
If it looks like a duck and quacks like a duck but needs batteries,
you probably have the wrong abstraction.

LSP

- **The "L" in SOLID is for *Liskov Substitution Principle,*** which states that sub-clases should be substitutable for the classes from which they were derived.
- Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it.
- For example, if MySubclass is a subclass of MyClass, you should be able to replace MyClass with MySubclass without bunging up the program.

LSP

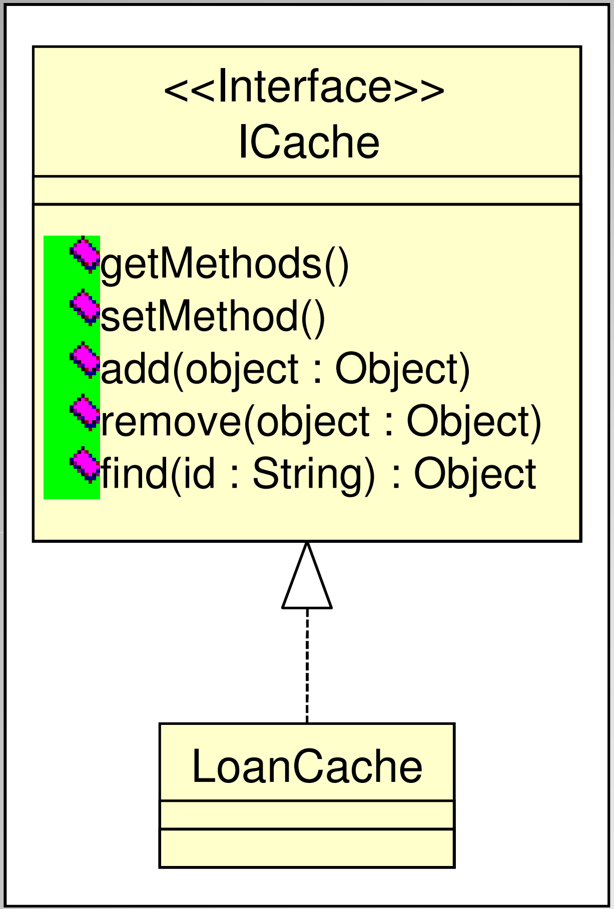
Interface Segregation Principle
You want me to plug this in *where?*

ISP

- **The “I” in SOLID is for *Interface Segregation Principle,*** which states that clients should not be forced to depend on methods they don’t use.

- If a class exposes so many members that those members can be broken down into groups that serve different clients that don’t use members from the other groups, you should think about exposing those member groups as separate interfaces.

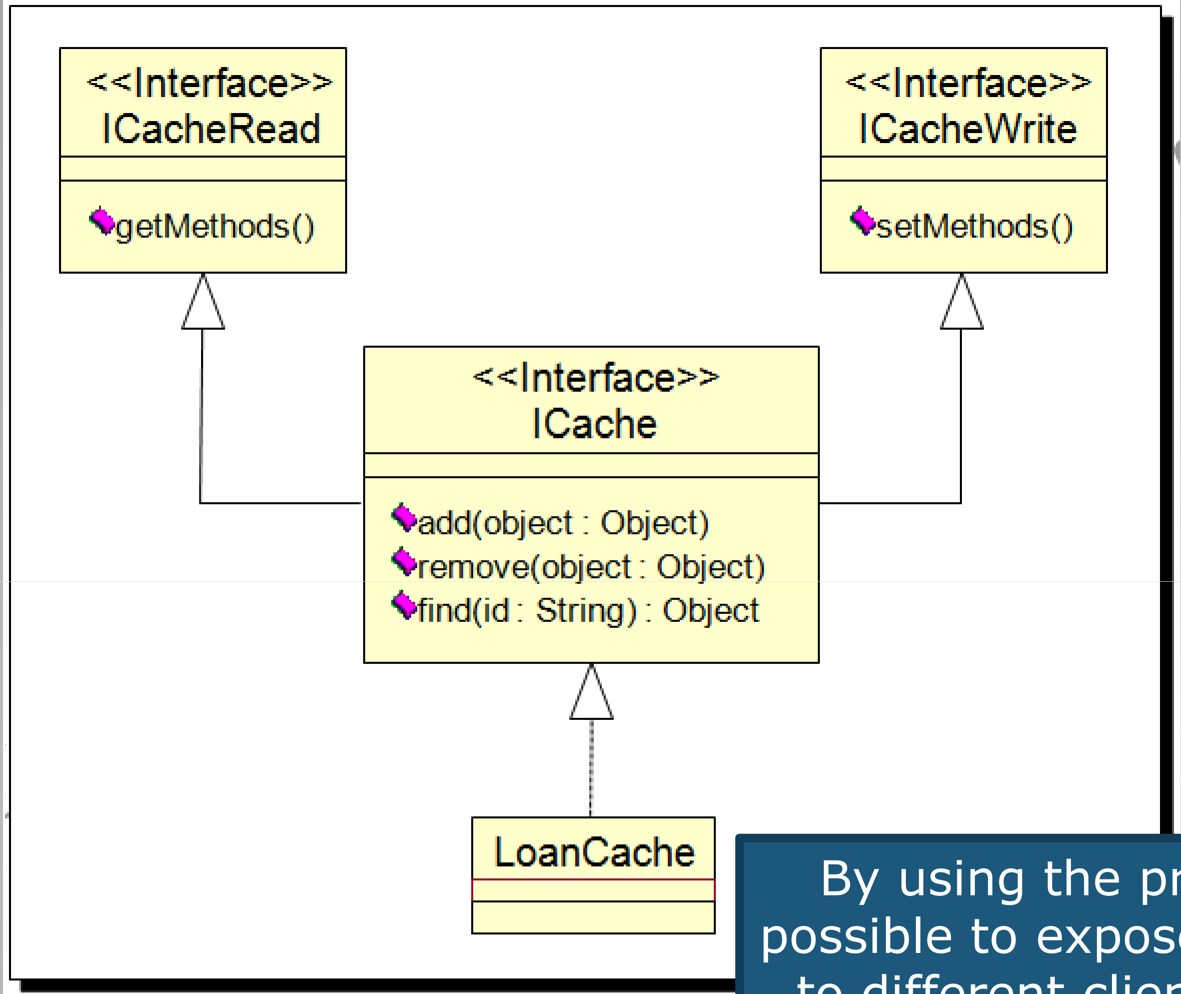
ISP



It is important to restrict one category of clients to only one set of operations

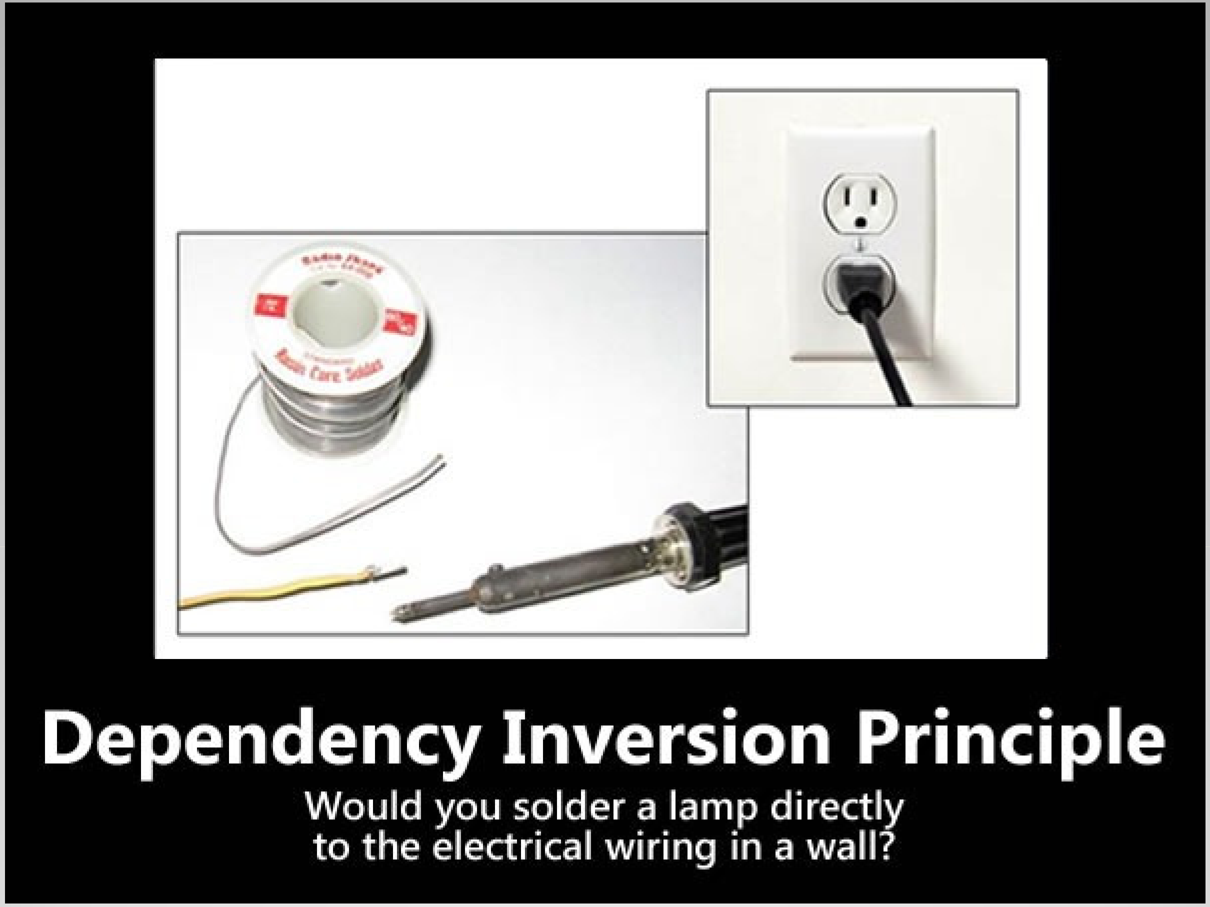
How do you ensure that different clients are still provided with an instance of the LoanCache but are restricted to using only limited set of operations?

ISP



By using the principle of ISP, it is possible to expose different interfaces to different clients and still present the same instance

LSP

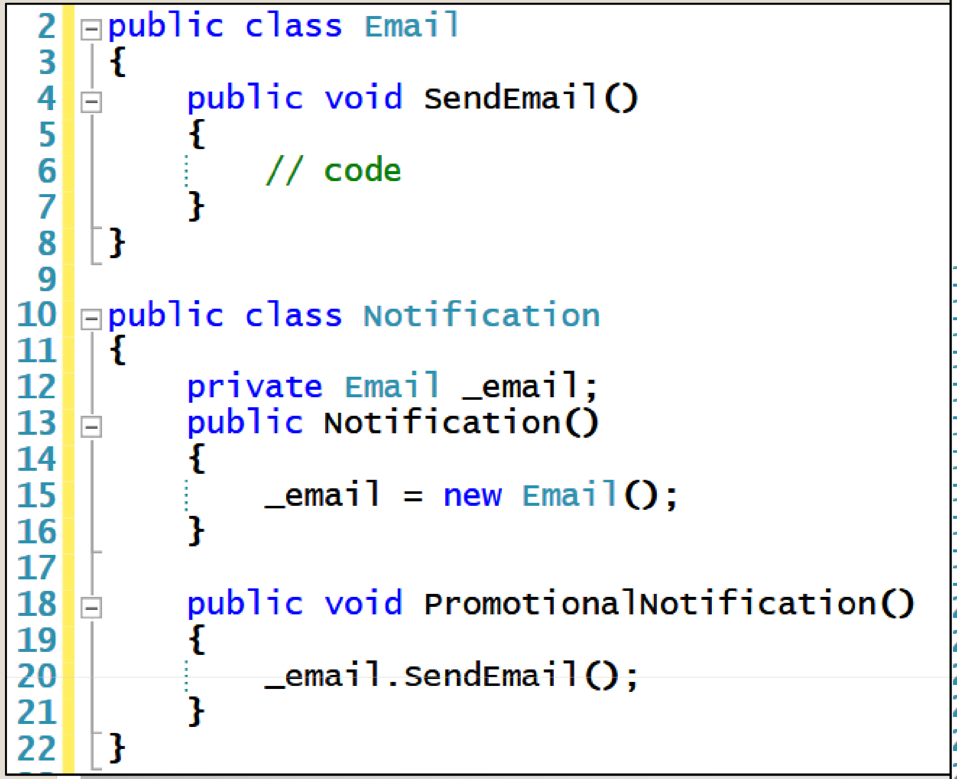

Dependency Inversion Principle

Would you solder a lamp directly
to the electrical wiring in a wall?

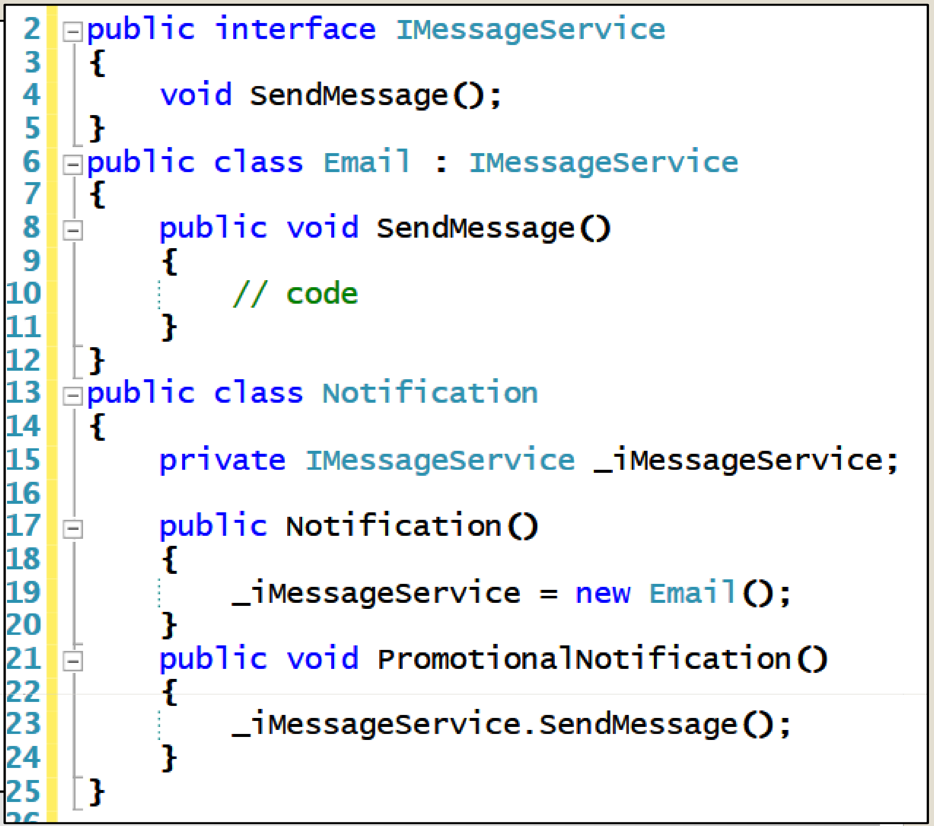
DIP

- **The “D” in SOLID is for *Dependency Inversion Principle*,** which states that high-level modules shouldn’t depend on low-level modules, but both should depend on shared abstractions.
- In addition, abstractions should not depend on details – instead, details should depend on abstractions.

DIP

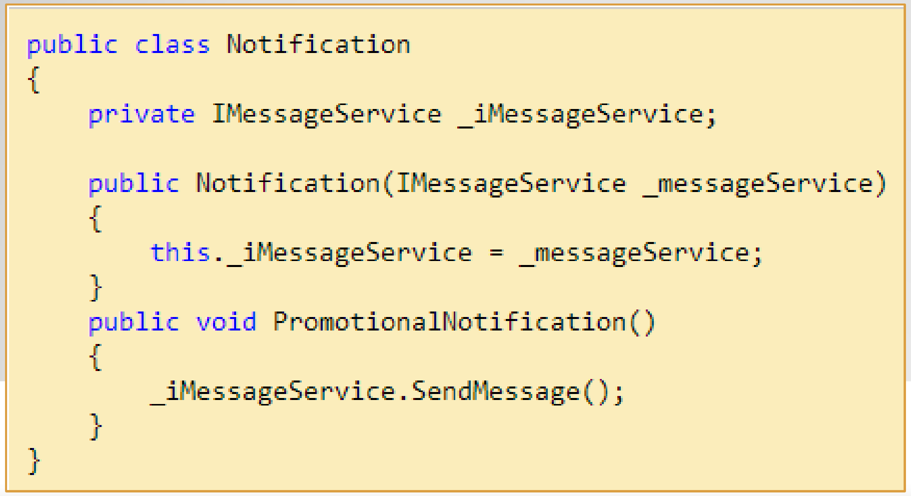


```csharp
public class Email
{
    public void SendEmail()
    {
        // code
    }
}

public class Notification
{
    private Email _email;
    public Notification()
    {
        _email = new Email();
    }

    public void PromotionalNotification()
    {
        _email.SendEmail();
    }
}
```

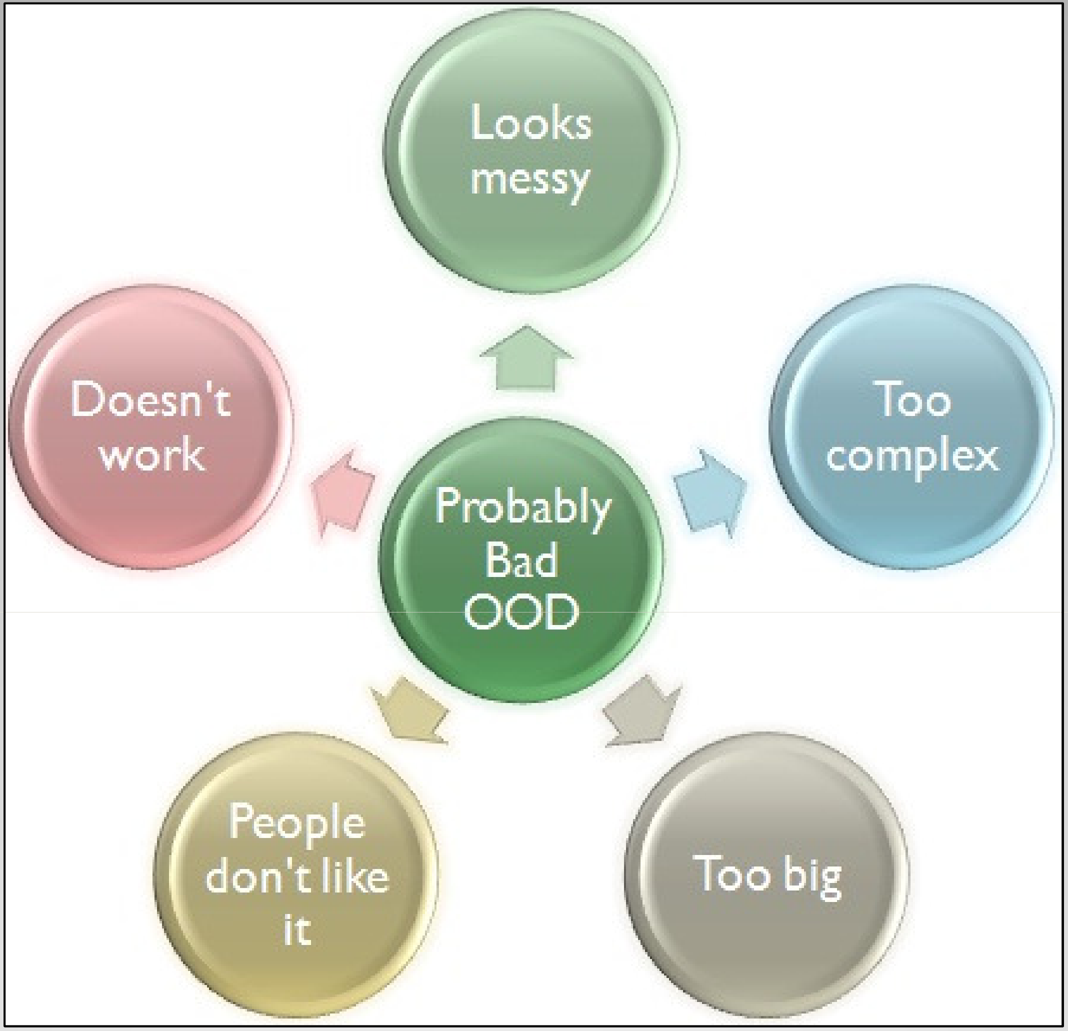


```csharp
public interface IMessageService
{
    void SendMessage();
}
public class Email : IMessageService
{
    public void SendMessage()
    {
        // code
    }
}
public class Notification
{
    private IMessageService _iMessageService;

    public Notification()
    {
        _iMessageService = new Email();
    }
    public void PromotionalNotification()
    {
        _iMessageService.SendMessage();
    }
}
```

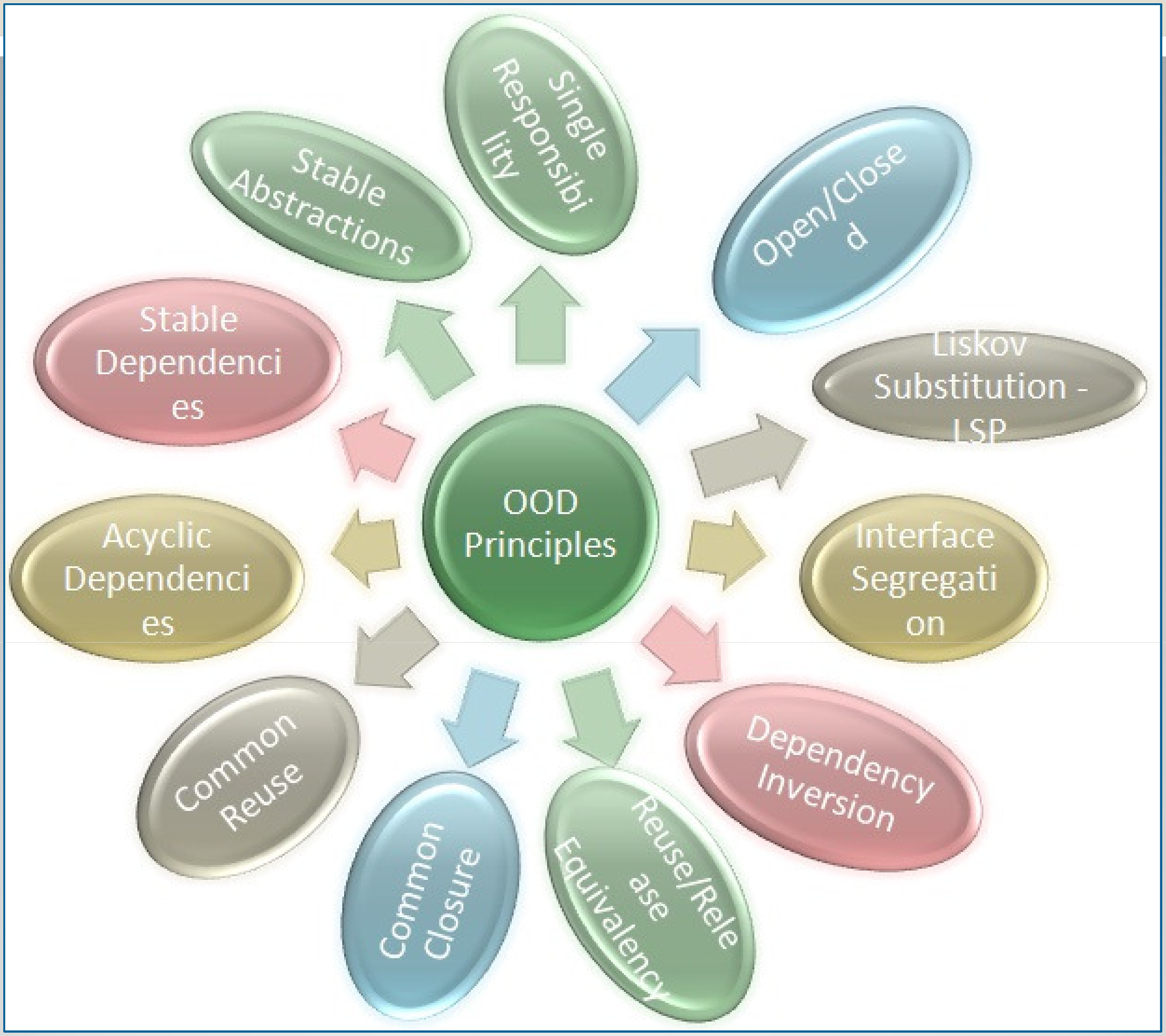
DIP



```csharp
public class Notification
{
    private IMessageService _iMessageService;

    public Notification(IMessageService _messageService)
    {
        this._iMessageService = _messageService;
    }
    public void PromotionalNotification()
    {
        _iMessageService.SendMessage();
    }
}
```



Rules for identifying BAD design



Not just S.O.L.I.D.....