Object Oriented Design Principles

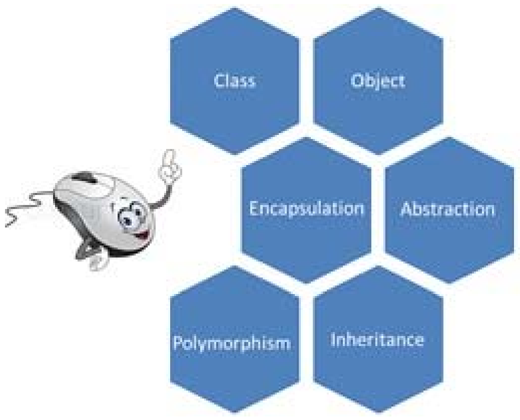
Marla Sukesh, 8 Apr 2013 CPOL

★★★★★ 4.92 (199 votes)

This article is intended for who have at least basic idea about Object oriented programming.

Who is Audience?

This article is intended for those who have at least a basic idea of Object oriented programming. They know the difference between classes and objects and can talk about the basic pillars of object oriented programming i.e., Encapsulation, Abstraction, Polymorphism and Inheritance.



Introduction

In the object oriented world we only see objects. Objects interact with each other. **Classes, Objects, Inheritance, Polymorphism, Abstraction** are common vocabulary we hear in our day-to-day careers.

In the modern software world every software developer uses object oriented language of some kind, but the question is, does he really know what object oriented programming means? Does he know that he is working as an object oriented programmer? If the answer is yes, is he really using the power of object oriented programming?

In this article we will go beyond the basic pillars of object oriented programming and talk about object oriented design.

Object Oriented Design

It’s a process of planning a software system where objects will interact with each other to solve specific problems. The saying goes, "Proper Object oriented design makes a developer's life easy, whereas bad design makes it a disaster."

How does anyone start?

When anyone starts creating software architecture their intentions are good. They try to use their existing experience to create an elegant and clean design.

Over time software starts to rot. With every feature request or change software design alters its shape and eventually the simplest changes to application requires a lot of effort and, more importantly, creates higher chances for more bugs.

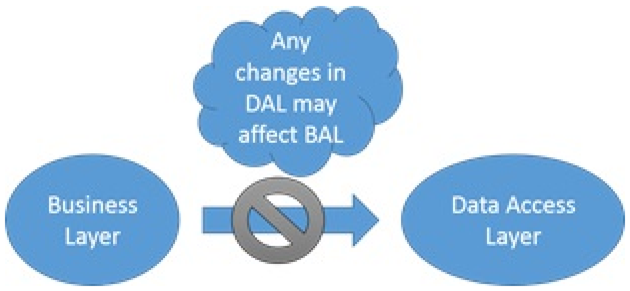
Who is to Blame

Software solves real life business problems and since business processes keep evolving, software keeps on changing.

Changes are an integral part of the software world. Obviously because clients are paying they will demand for what they are expecting. So we cannot blame "change" for the degradation of software design. It is our design which is at fault.

One of the biggest reasons for the damaging of software design is the introduction of unplanned dependencies into the system. Every part of the system is dependant on some other part and so changing one part will affect another part. If we are able to manage those dependencies we will easily maintain the software system and software quality too.

Example



Solution - Principles, Design Patterns and Software architecture

- **Software architecture** like MVC, 3-Tier, MVP tells us how overall projects are going to be structured.
- **Design pattern** allows us to reuse the experience or rather, provides reusable solutions to commonly occurring problems. Example – an object creation problem, an instance management problem, etc.
- **Principles** tell us, do these and you will achieve this. How you will do it is up to you. Everyone defines some principles in their lives like, "I never lie," or, "I never drink alcohol," etc. He/she follow these principles to make his/her life easy, but how will he/she stick to these principles is up to the individual.

In the same way, object oriented design is filled with many principles which let us manage the problems with software design.

Mr. Robert Martin (commonly known as Uncle Bob) categorized them as

1. Class Design principles – Also called SOLID
2. Package Cohesion Principles
3. Package Coupling principle

In this article we will talk about SOLID principles with practical example.

SOLID

It's an acronym of ☺ five principles introduced by **Mr. Robert Martin** (commonly known as Uncle Bob) i.e., Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversion. It's said (Wikipedia) when all five principles are applied together intend to make it more likely that a programmer will create a system that is easy to maintain and extend over time. Let's talk about every principle in detail

I) S - SRP - Single responsibility Principle

Real world comparison

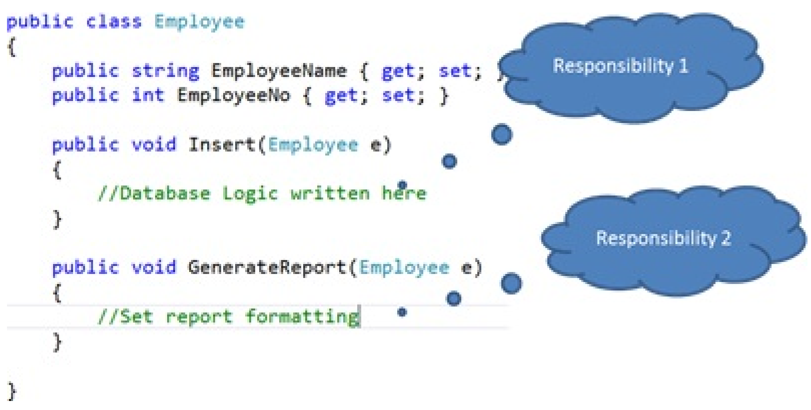
S/W engineer
Author
News paper editor
CAN'T YOU DO ANYTHING RIGHT?
Copyright 2003 Randy Glasbergen. www.glasbergen.com

I work as a team leader for one of the software firms in India. In my spare time I do some writing, newspaper editing and other various projects. Basically, I have multiple responsibilities in my life.

When something bad happens at my work place, like when my boss scolds me for some mistake, I get distracted from my other work. Basically, if one thing goes bad, everything will mess up.

Identify Problem in Programming

Before we talk about this principle I want you take a look at following class.



```
public class Employee
{
    public string EmployeeName { get; set; }
    public int EmployeeNo { get; set; }

    public void Insert(Employee e)
    {
        //Database Logic written here
    }

    public void GenerateReport(Employee e)
    {
        //Set report formatting
    }

}
```

Every time insert logic changes, this class will change.
Every time report format changes, this class will changes.

What is the issue?

Every time one gets changed there is a chance that the other also gets changed because both are staying in the same home and both have same parent. We can't control everything. So a single change leads to double testing (or maybe more).

What is SRP?

SRP says "Every software module should have only one reason to change".

- Software Module – Class, Function etc.
- Reason to change - Responsibility

Solutions which will not Violate SRP

Now it's up to us how we achieve this. One thing we can do is create three different classes

1. `Employee` – Contains Properties (Data)
2. `EmployeeDB` – Does database operations
3. `EmplyeeReport` – Does report related tasks

```
public class Employee
{
    public string EmployeeName { get; set; }
    public int EmployeeNo { get; set; }
}
public class EmployeeDB
{
    public void Insert(Employee e)
    {
        //Database Logic written here
    }
 public Employee Select()
    {
        //Database Logic written here
    }
}
public class EmployeeReport
{
    public void GenerateReport(Employee e)
    {
        //Set report formatting
    }
}
```

Note: This principle also applies to methods. Every method should have a single responsibility.

Can a single class can have multiple methods?

The answer is YES. Now you might ask how it's possible that

1. A class will have single responsibility.
2. A method will have single responsibility.
3. A class may have more than one method.

Well the answer for this question is simple. It's context. Here, responsibility is related to the context in which we are speaking. When we say class responsibility it will be somewhat at higher level. For instance, the `EmployeeDB` class will be responsible for employee operations related to the Database whereas the `EmployeeReport` class will be responsible for employee operations related to reports.

When it comes to methods it will be at lower level. For instance look at following example:

```
//Method with multiple responsibilities – violating SRP
public void Insert(Employee e)
{
    string StrConnectionString = "";
    SqlConnection objCon = new SqlConnection(StrConnectionString);
    SqlParameter[] SomeParameters=null;//Create Parameter array from values
    SqlCommand objCommand = new SqlCommand("InertQuery", objCon);
    objCommand.Parameters.AddRange(SomeParameters);
    ObjCommand.ExecuteNonQuery();
}

//Method with single responsibility - follow SRP
public void Insert(Employee e)
{
    SqlConnection objCon = GetConnection();
    SqlParameter[] SomeParameters=GetParameters();
    SqlCommand ObjCommand = GetCommand(objCon,"InertQuery",SomeParameters);
    ObjCommand.ExecuteNonQuery();
}

private SqlCommand GetCommand(SqlConnection objCon, string InsertQuery, SqlParameter[]
SomeParameters)
{
    SqlCommand objCommand = new SqlCommand(InsertQuery, objCon);
    objCommand.Parameters.AddRange(SomeParameters);
    return objCommand;
}

private SqlParameter[] GetParaeters()
{
    //Create Paramter array from values
}

private SqlConnection GetConnection()
{
    string StrConnectionString = "";
    return new SqlConnection(StrConnectionString);
}
```

Testing is advantageous in and of itself, but that the code has become readable is an additional advantage. The more the code is readable the simpler it seems.

II) O - OCP – Open Close Principle

Real World Comparison

Let's assume you want to add one more floor between the first and second floor in your two floor house. Do you think it is possible? Yes it is, but is it feasible? Here are some options:

- One thing you could have done at time you were building the house first time was make it with three floors, keeping second floor empty. Then utilize the second floor anytime you want. I don't know how feasible that is, but it is one solution.
- Break the current second floor and build two new floors, which is not sensible.

Identify Problem in Programming

Let's say the `Select` method in the `EmployeeDB` class is used by two clients/screens. One is made for normal employees, one is made for managers, and the Manager Screen needs a change in the method.

If I make changes in the `Select` method to satisfy the new requirement, other UI will also be affected. Plus making changes in existing tested solution may result in unexpected errors.

What is OCP?

It says, "Software modules should be closed for modifications but open for extensions." An orthogonal statement.

Solution which will not violate OCP

1) Use of inheritance

We will derive a new class called `EmployeeManagerDB` from `EmployeeDB` and override the `Select` method as per the new requirement.

```
public class EmployeeDB
{
    public virtual Employee Select()
    {
        //Old Select Method
    }
}
public class EmployeeManagerDB : EmployeeDB
{
    public override Employee Select()
    {
        //Select method as per Manager
        //UI requirement
    }
}
```

Note: Now the design is considered good object oriented design if this change is anticipated at the time of the design and already has a provision within for extension (method made virtual). Now the UI code will look like:

```
//Normal Screen
EmployeeDB objEmpDb = new EmployeeDB();
Employee objEmp = objEmpDb.Select();

//Manager Screen
EmployeeDB objEmpDb = new EmployeeManagerDB();
Employee objEmp = objEmpDb.Select();
```

2) Extension method

If you are using .NET 3.5 or later then there is a second way called extension method that will let us add new methods to existing types without even touching them.

Note: There may be some more ways to achieve the desired result. As I said these are principles not commandments.

III) L – LSP – Liskov substitution principle

What is LSP?

You might be wondering why we are defining it prior to examples and problem discussions. Simply put, I thought it will be more sensible here.

It says, "Subclasses should be substitutable for base classes." Don't you think this statement is strange? If we can always write `BaseClass b=new DerivedClass()` then why would such a principle be made?

Real World Comparison

A father is a real estate business man whereas his son wants to be cricketer.

A son can't replace his father in spite of the fact that they belong to same family hierarchy.

Identify Problem in Programming

Let's talk about a very common example.

Normally when we talk about geometric shapes, we call a rectangle a base class for square. Let's take a look at code snippet.

```
public class Rectangle
{
    public int Width { get; set; }
    public int Height { get; set; }
}

public class Square:Rectangle
{
    //codes specific to
    //square will be added
}
```

One can say,

Hide   Copy Code

```
Rectangle o = new Rectangle();
o.Width = 5;
o.Height = 6;
```

Perfect but as per LSP we should be able to replace Rectangle with square. Let's try to do so.

```
Rectangle o = new Square();
o.Width = 5;
o.Height = 6;
```

What is the matter? Square cannot have different width and height.

What it means? It means we can't replace base with derived. Means we are violating LSP.

Why don't we make width and height virtual in Rectangle, and override them in Square?

```
public class Square : Rectangle
{
    public override int Width
    {
        get{return base.Width;}
        set
        {
            base.Height = value;
            base.Width = value;
        }
    }
    public override int Height
    {
        get{return base.Height;}
        set
        {
            base.Height = value;
            base.Width = value;
        }
    }
}
```

We can't because doing so we are violating LSP, as we are changing the behavior of Width and Height properties in derived class (for Rectangle height and width cannot be equal, if they are equal it's cannot be Rectangle).

(It will not be a kind of replacement).

Solution which will not violate LSP

There should be an abstract class Shape which looks like:

```
public abstract class Shape
{
    public virtual int Width { get; set; }
    public virtual int Height { get; set; }
}
```

Now there will be two concrete classes independent of each other, one rectangle and one square, both of which will be derived from Shape.

Now the developer can say:

```
Shape o = new Rectangle();
o.Width = 5;
o.Height = 6;

Shape o = new Square();
o.Width = 5; //both height and width become 5
o.Height = 6; //both height and width become 6
```

Even after overriding in derived classes we are not changing the behavior of width and height, because when we talk about shape, there will not be any fixed rule for width and height. They may be equal, or may not be.

IV) I – ISP– Interface Segregation principle

Real World Comparison

Let’s say you purchase a new desktop PC. You will find a couple of USB ports, some serial ports, a VGA port etc. If you open the cabinet you will see lots of slots on the motherboard used for connecting various parts with each other, mostly used by hardware engineers at the time of assembly.

Those internal slots will not be visible until you open the cabinet. In short, only the required interfaces are made available/visible to you. Imagine a situation where everything was external or internal. Then there is a greater chances of hardware failure (as if life wasn't hard enough for computer users).

Let's say we will go to a shop to buy something (let's say, for instance, to buy a cricket bat).

Now imagine a situation where the shopkeeper starts showing you the ball and stumps as well. It may be possible that we will get confused and may end up buying something we did not require. We may even forget why we were there in the first place.

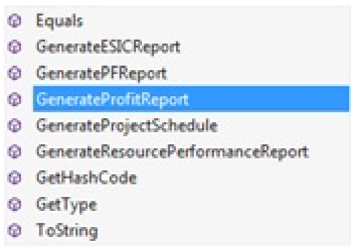
Identify Problem in Programming

Let's say we want to develop a Report Management System. Now, the very first task is creating a business layer which will be used by three different UIs.

1. `EmployeeUI` – Show reports related to currently logged in employee
2. `ManagerUI` – Show reports related to himself and the team for which he/manager belongs.
3. `AdminUI` – Show reports related to individual employee ,related to team and related to company like profit report.

```
public interface IReportBAL
{
    void GeneratePFReport();
    void GenerateESICReport();

    void GenerateResourcePerformanceReport();
    void GenerateProjectSchedule();

    void GenerateProfitReport();
}
public class ReportBAL : IReportBAL
{
    public void GeneratePFReport()
    {/*...............*/}

    public void GenerateESICReport()
    {/*...............*/}

    public void GenerateResourcePerformanceReport()
    {/*..............*/}

    public void GenerateProjectSchedule()
    {/*..............*/}

    public void GenerateProfitReport()
    {/*..............*/}
}
public class EmployeeUI
{
    public void DisplayUI()
    {
        IReportBAL objBal = new ReportBAL();
        objBal.GenerateESICReport();
        objBal.GeneratePFReport();
    }
}
public class ManagerUI
{
    public void DisplayUI()
    {
        IReportBAL objBal = new ReportBAL();
        objBal.GenerateESICReport();
        objBal.GeneratePFReport();
        objBal.GenerateResourcePerformanceReport ();
        objBal.GenerateProjectSchedule ();
    }
}
public class AdminUI
{
    public void DisplayUI()
    {
        IReportBAL objBal = new ReportBAL();
        objBal.GenerateESICReport();
        objBal.GeneratePFReport();
        objBal.GenerateResourcePerformanceReport();
        objBal.GenerateProjectSchedule();
        objBal.GenerateProfitReport();
    }
}
```

Now in each UI when the developer types "`objBal`" the following intellisense will be shown:



```
Equals
GenerateESICReport
GeneratePFReport
GenerateProfitReport
GenerateProjectSchedule
GenerateResourcePerformanceReport
GetHashCode
GetType
ToString
```

What is the problem?

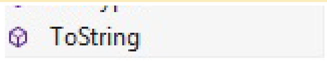
The developer who is working on `EmployeeUI` gets access to all the other methods as well, which may unnecessarily cause him/her confusion.

What is ISP?

It states that "Clients should not be forced to implement interfaces they don’t use." It can also be stated as "Many client specific interfaces are better than one general purpose interface." In simple words, if your interface is fat, break it into multiple interfaces.

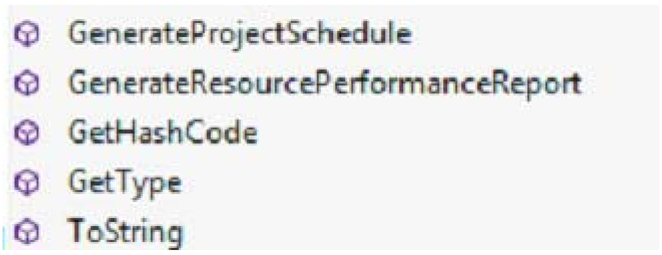
Update code to follow ISP

```
public interface IEmployeeReportBAL
{
    void GeneratePFReport();
    void GenerateESICReport();
}
public interface IManagerReportBAL : IEmployeeReportBAL
{
    void GenerateResourcePerformanceReport();
    void GenerateProjectSchedule();
}
public interface IAdminReportBAL : IManagerReportBAL
{
    void GenerateProfitReport();
}
public class ReportBAL : IAdminReportBAL
{
    public void GeneratePFReport()
    {/*...............*/}

    public void GenerateESICReport()
    {/*...............*/}

    public void GenerateResourcePerformanceReport()
    {/*...............*/}

    public void GenerateProjectSchedule()
    {/*...............*/}

    public void GenerateProfitReport()
    {/*...............*/}
}
```

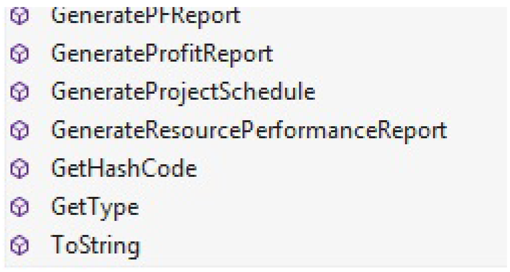


```
public class EmployeeUI
```

```
{
    public void DisplayUI()
    {
        IEmployeeReportBAL objBal = new ReportBAL();
        objBal.GenerateESICReport();
        objBal.GeneratePFReport();
    }
}
```


GenerateProjectSchedule
GenerateResourcePerformanceReport
GetHashCode
GetType
ToString

```
public class ManagerUI
{
    public void DisplayUI()
    {
        IManagerReportBAL  objBal = new ReportBAL();
        objBal.GenerateESICReport();
        objBal.GeneratePFReport();
        objBal.GenerateResourcePerformanceReport ();
        objBal.GenerateProjectSchedule ();
    }
}
```


GeneratePFReport
GenerateProfitReport
GenerateProjectSchedule
GenerateResourcePerformanceReport
GetHashCode
GetType
ToString

```
public class AdminUI
{
    public void DisplayUI()
    {
        IAdminReportBAL  objBal = new ReportBAL();
        objBal.GenerateESICReport();
        objBal.GeneratePFReport();
        objBal.GenerateResourcePerformanceReport();
        objBal.GenerateProjectSchedule();
        objBal.GenerateProfitReport();
    }
}
```

By following ISP we can make client see, what he is required to see.

V) D – DIP– Dependency Inversion principle

Real World Comparison

Let's talk about our desktop computers. Different parts such as RAM, a hard disk, and CD-ROM (etc.) are loosely connected to the motherboard. That means that, if, in future in any part stops working it can easily be replaced with a new one. Just imagine a situation where all parts were tightly coupled to each other, which means it would not be possible to remove any part from the motherboard.

Then in that case if the RAM stops working we have to buy new motherboard which is going to be very expensive.

Identify Problem in Programming

Look at the following code.

```
public class CustomerBAL
{
    public void Insert(Customer c)
    {
        try
        {
            //Insert logic
        }
        catch (Exception e)
        {
            FileLogger f = new FileLogger();
            f.LogError(e);
        }
    }
}

public class FileLogger
{
    public void LogError(Exception e)
    {
        //Log Error in a physical file
    }
}
```

In the above code `CustomerBAL` is directly dependent on the `FileLogger` class which will log exceptions in physical file. Now let's assume tomorrow management decides to log exceptions in the Event Viewer. Now what? Change existing code. Oh no! My God, that might create a new error!

What is DIP?

It says, "High level modules should not depend upon low level modules. Rather, both should depend upon abstractions."

Solution with DIP

Hide Shrink ▲ Copy Code

```
public interface ILogger
```

```
{
    void LogError(Exception e);
}

public class FileLogger:ILogger
{
    public void LogError(Exception e)
    {
        //Log Error in a physical file
    }
}
public class EventViewerLogger : ILogger
{
    public void LogError(Exception e)
    {
        //Log Error in a physical file
    }
}
public class CustomerBAL
{
    private ILogger _objLogger;
    public CustomerBAL(ILogger objLogger)
    {
        _objLogger = objLogger;
    }

    public void Insert(Customer c)
    {
        try
        {
            //Insert logic
        }
        catch (Exception e)
        {
            _objLogger.LogError(e);
        }
    }
}
```

As you can see the client depends on abtraction i.e, `ILogger` which can be set to an instance of any derived class.

So now we've covered all five principles of SOLID. Thanks Uncle Bob.

Is it end?

Now the question is, are there more principles other than those categorized by Uncle Bob? The answer is yes, but we will not going to describe each and every thing in detail for now. But they are:

- Program to Interface Not Implementation.
- Don't Repeat Yourself.
- Encapsulate What Varies.
- Depend on Abstractions, Not Concrete classes.
- Least Knowledge Principle.
- Favor Composition over Inheritance.
- Hollywood Principle.
- Apply Design Pattern wherever possible.
- Strive for Loosely Coupled System.
- Keep it Simple and Sweet / Stupid.

Conclusion

We can't avoid changes. The only thing we can do is develop and design software in such a way that it is able to handle such changes.

- SRP should be kept in mind while creating any class, method or any other module (which even applies to SQL stored procedures and functions). It makes code more readable, robust, and testable.
- As per my experience we can't follow DIP each and every time, sometimes we have to depend on concrete classes. The only thing we have to do is understand the system, requirements and environment properly and find areas where DIP should be followed.
- Following DIP and SRP will opens a door to implement OCP as well.
- Make sure to create specific interfaces so that complexities and confusions will be kept away from end developers, and thus, the ISP will not get violated.
- While using inheritance take care of LSP.

Hope all of you enjoyed reading this article. Thank you for the patience.

For technical training related to various topics including ASP.NET, Design Patterns, WCF and MVC contact SukeshMarla[at]Gmail.com or at www.sukesh-marla.com

For more stuff like this click here. Subscribe to article updates or follow at twitter @SukeshMarla

See 400 above FAQ questions and answers in .NET, C#, ASP.NET, SQL, WCF, WPF, WWF, SharePoint, Design patterns, UML etc.

License

About the Author

Marla Sukesh

Instructor / Trainer Train IT
India

Learning is fun but teaching is awesome.

Code re-usability is my passion ,Teaching and learning is my hobby, Becoming an successful entrepreneur is my goal.

By profession I am a Corporate Trainer.
I do trainings on WCF, MVC, Business Intelligence, Design Patterns, HTML 5, jQuery, JSON and many more Microsoft and non-Micrsoft technologies.
Find my profile here
My sites

- **justcompile.com**
- **www.sukesh-marla.com**