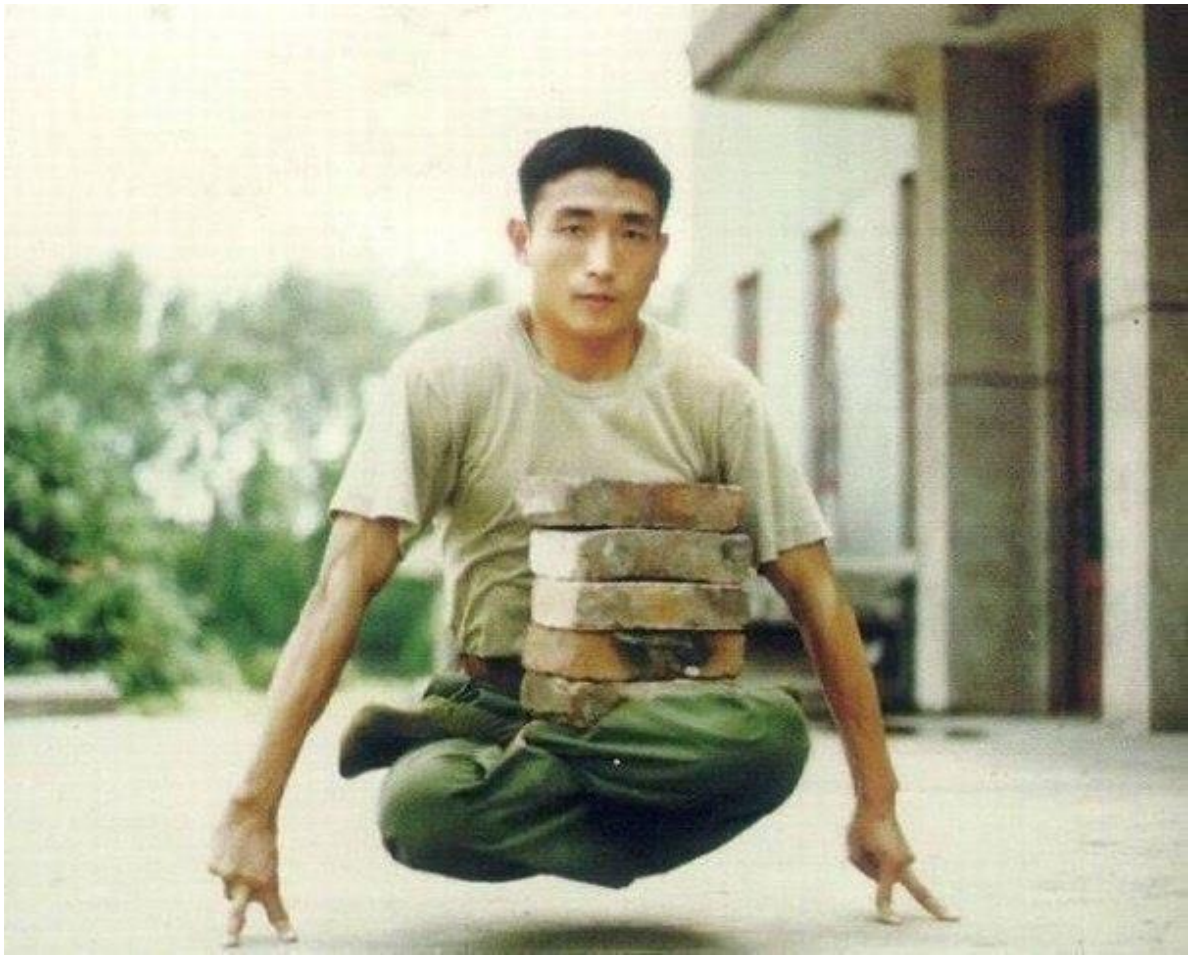
Good coding practice in real life

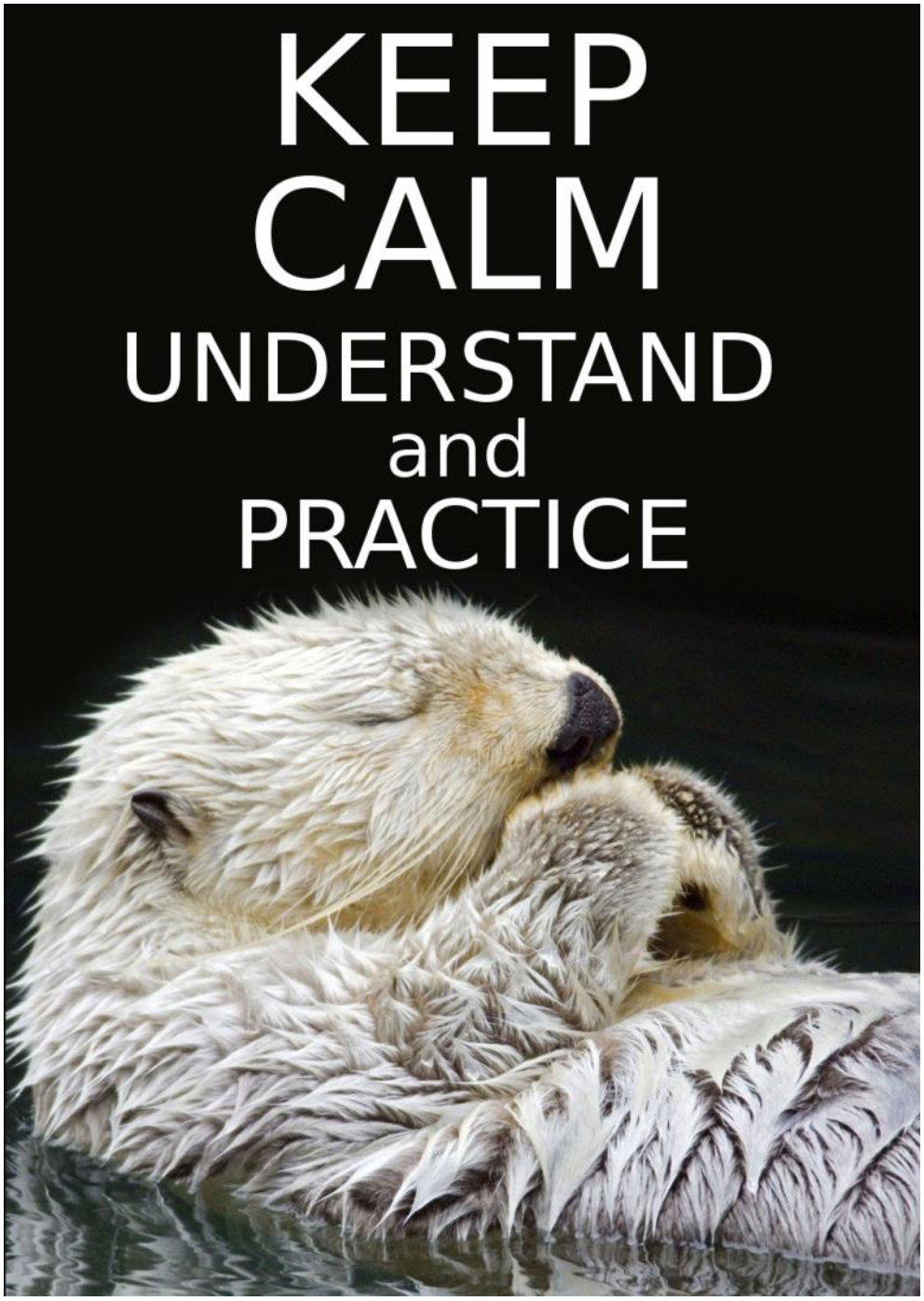
(with a focus on DIP)

Vladimir Alekseichenko

Good practices make life easier

Good practices are not easy




KEEP
CALM
UNDERSTAND
and
PRACTICE

Agenda

- The dependency in software.
- Mainstream and depth understanding of management dependencies.
- Short history about closely connected principles.
- The abstraction.
- The dependency inversion principle.
- Some examples of DIP violations and solutions for each of them.

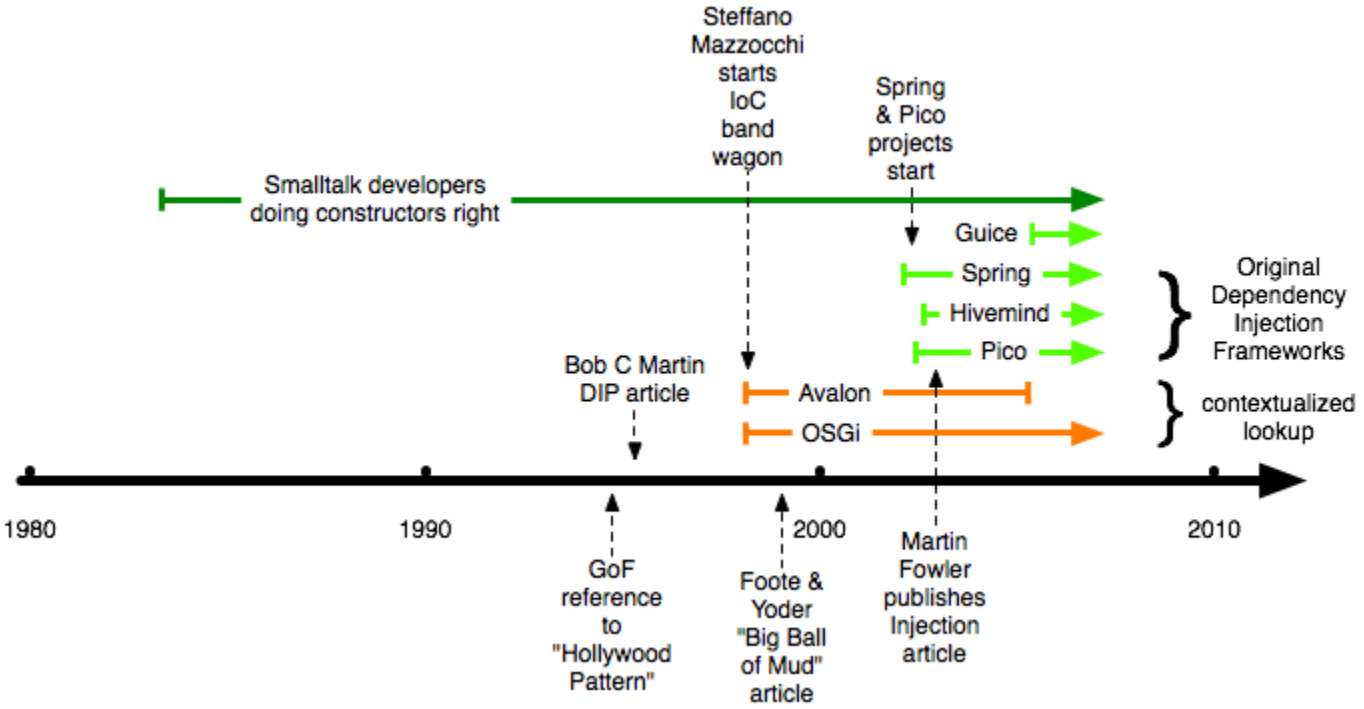
What is it?

- Hollywood principle
- Dependency inversion
- Inversion of control
- IoC container
- Service locator
- Dependency injection
- Constructor injection
- Setter injection
- Interface injection

How do we handle this?

- Which is better?
- What these solutions have in common?
- Which we have to use in real projects?

History



http://picocontainer.org/inversion-of-control-history.html

Hollywood principle

Don't call us, we'll call you!

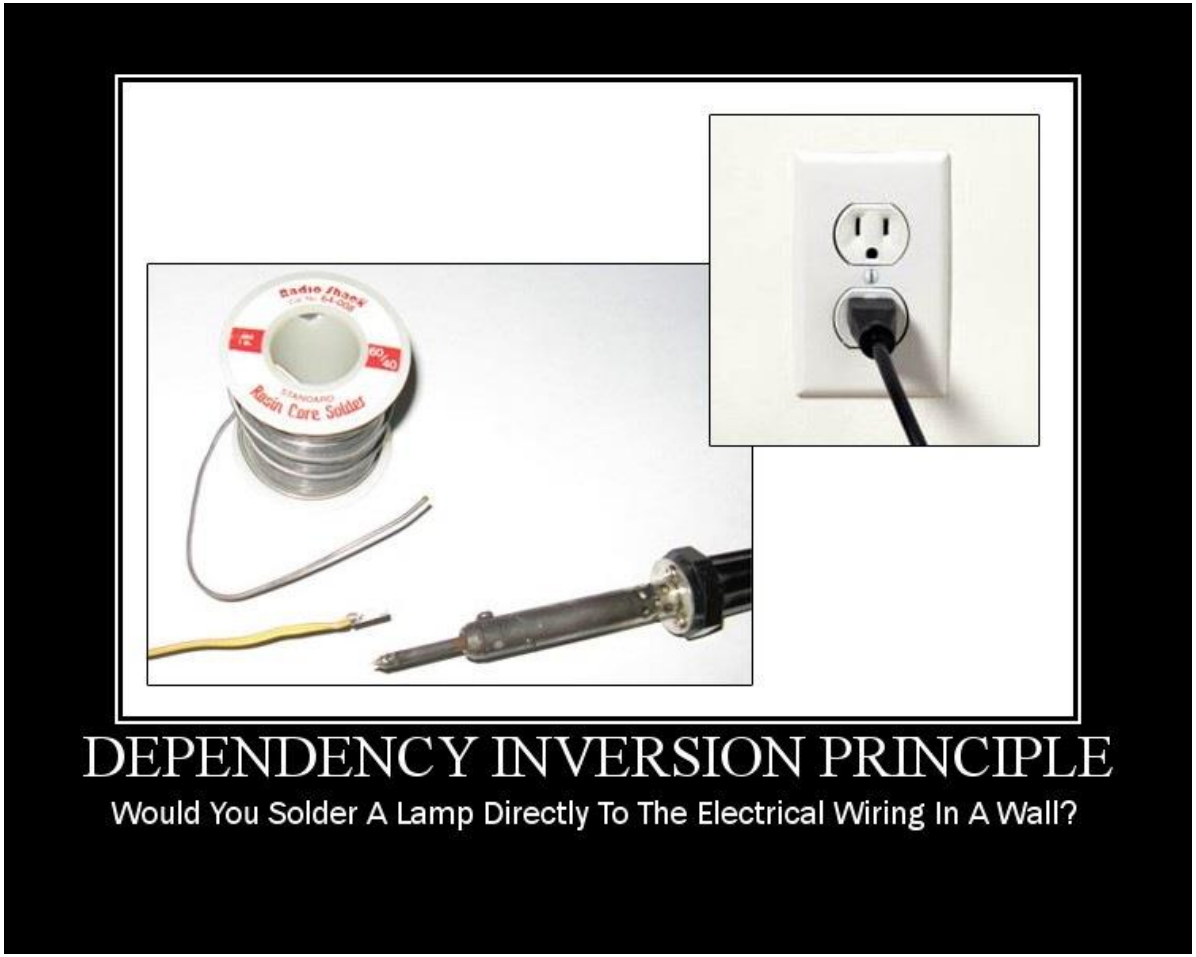
Inversion of Control

Three things:

- **Component dependencies**
- Configuration
- Component lifecycle

http://picocontainer.org/lifecycle.html

Dependency inversion principle



Dependency inversion principle

The **dependency inversion principle** was postulated by **Robert C. Martin** and described in several publications.

Robert Cecil Martin aka "Uncle Bob"

http://www.butunclebob.com/CleanCoder.UncleBob

The Definition of a “Bad Design”

A piece of software that fulfills its requirements and yet exhibits any or all of the following three traits has a **bad design**.

- Rigidity
- Fragility
- Immobility

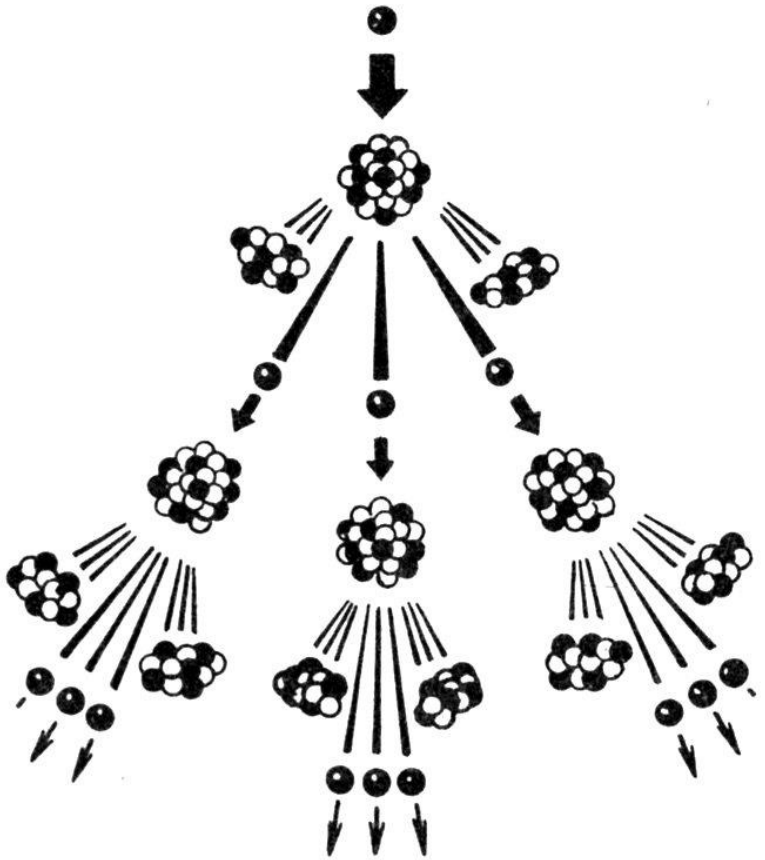
http://www.objectmentor.com/resources/articles/dip.pdf

Rigidity

Rigidity is due to the fact that a single change to heavily interdependent software begins a cascade of changes in dependent modules.

http://www.objectmentor.com/resources/articles/dip.pdf

Chain reaction



http://do.gendocs.ru/docs/index-210661.html?page=3

Nuclear mushroom

http://www.fdwallpapers.com/desktop.php?pid=2967

Fragility

Fragility is the tendency of a program to break in many places when a single change is made.

Often the new problems are in areas that have no conceptual relationship with the area that was changed.

http://www.objectmentor.com/resources/articles/dip.pdf

The art of rock balancing

Michael Grab
http://www.gravityglue.com/

Immobility

A design is immobile when the desirable parts of the design are highly dependent upon other details that are not desired.

http://www.objectmentor.com/resources/articles/dip.pdf

Card stacking

Bryan Berg
http://www.cardstacker.com

Why does software becomes **Rigid**, **Immobile**, **Fragile**?

http://siriusbuzz.com/what-happens-with-that-sirius-lifetime-subscription.php/big-question-mark

Improper
dependencies
between modules.

The **dependency inversion** principle consists of two parts.

The first part of the principle

High level modules should not depend upon low level modules. Both should depend upon abstractions.

http://www.objectmentor.com/resources/articles/dip.pdf

The second part of the principle

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

http://www.objectmentor.com/resources/articles/dip.pdf

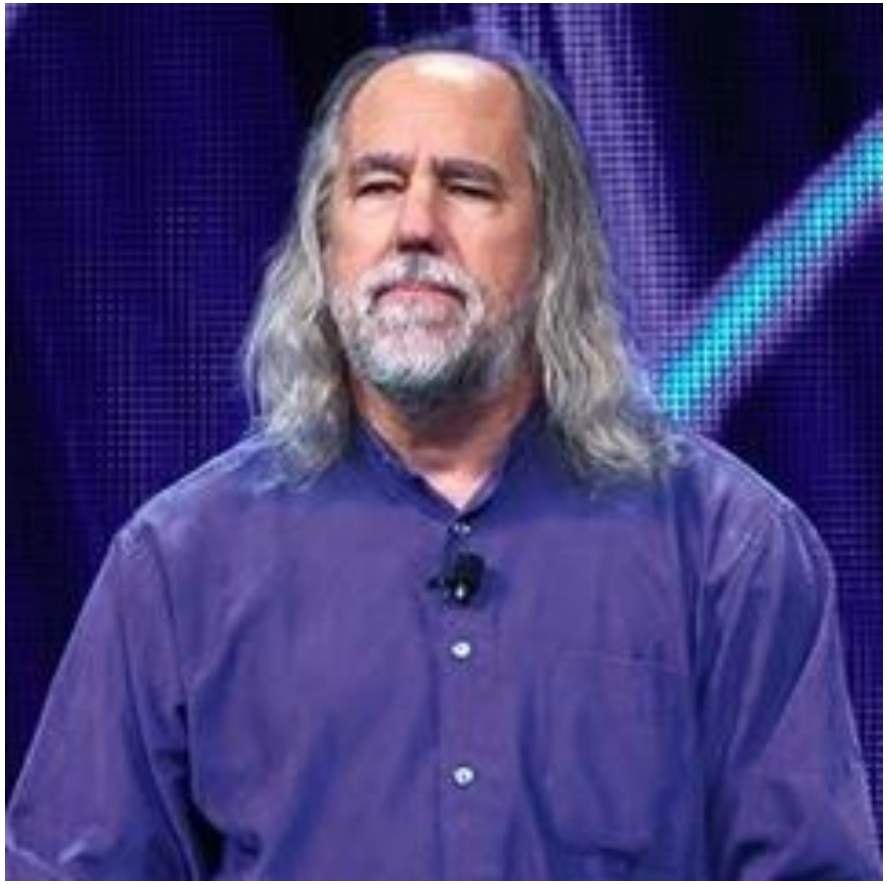
Dependency inversion principle

Don’t depend upon volatile concrete classes.

UML for Java (TM) Programmers by *Robert C. Martin.*

What is abstraction?

Grady Booch

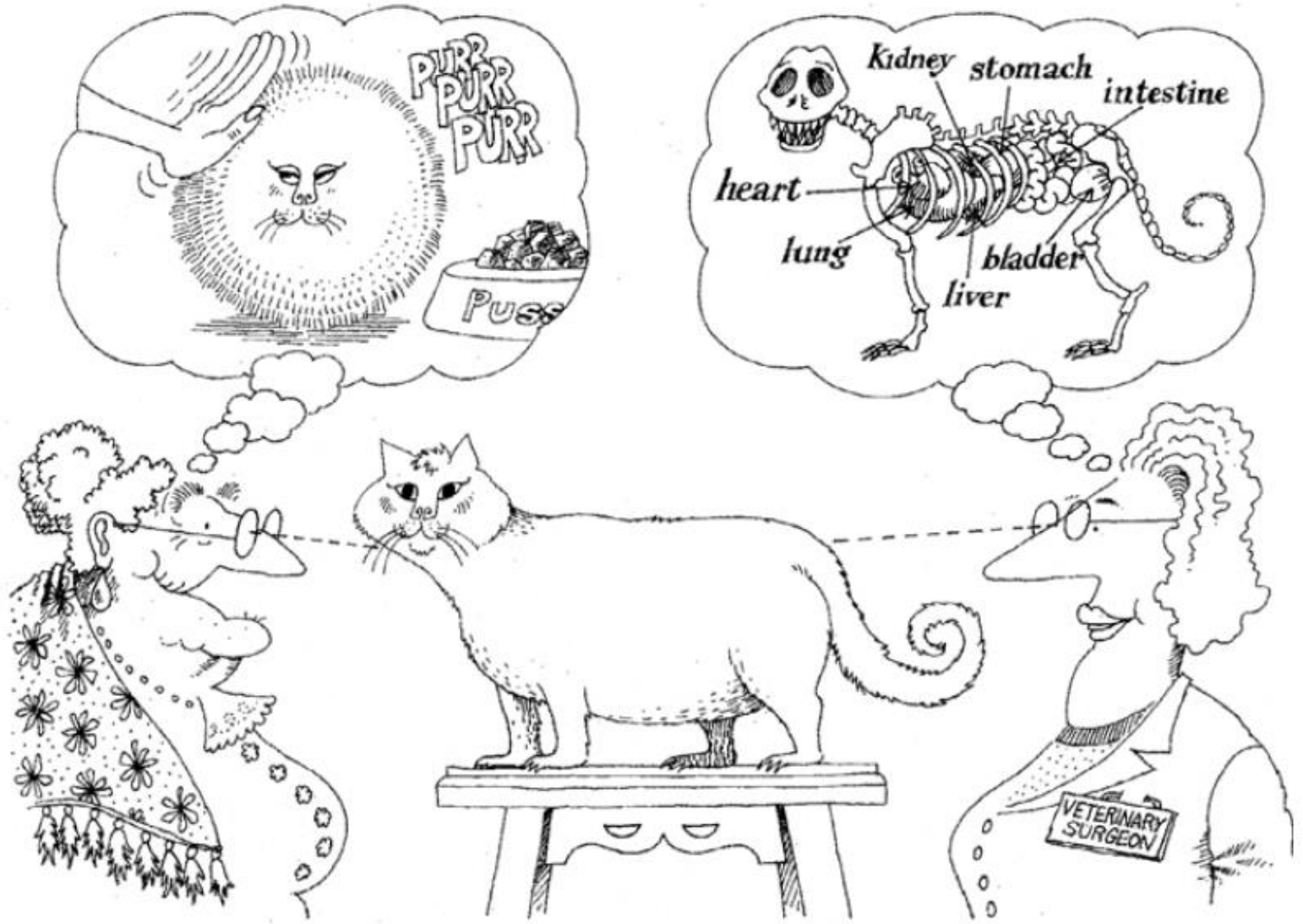


https://twitter.com/Grady_Booch

Abstraction

An **abstraction** denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer.

Object-Oriented Analysis and Design with Applications by *Grady Booch*



Abstraction focuses on the essential characteristics of some object, relative to the perspective of the viewer.

Dependency inversion principle

- If you **inherit** from a class, make it an **abstract** class.
- If you hold a **reference** to a class, make it an **abstract** class.
- If you call a **function**, make it an **abstract** function.

UML for Java (TM) Programmers by *Robert C. Martin.*

Dependency inversion principle

- In general, **abstract classes** and interfaces **change far less** often than their concrete derivatives.
- Therefore we would rather depend upon the abstractions than the concretions.
- Following this principle reduces the impact that a change can have upon the system.

UML for Java (TM) Programmers by *Robert C. Martin.*

Dependency inversion principle

Does this mean we can't use **string**, **int**…?

string *alias* **System.String**

int *alias* **System.Int32**

After all, they are **concrete classes**.

UML for Java (TM) Programmers by *Robert C. Martin.*

Does using them constitute a violation of DIP?

http://siriusbuzz.com/what-happens-with-that-sirius-lifetime-subscription.php/big-question-mark

Dependency inversion principle

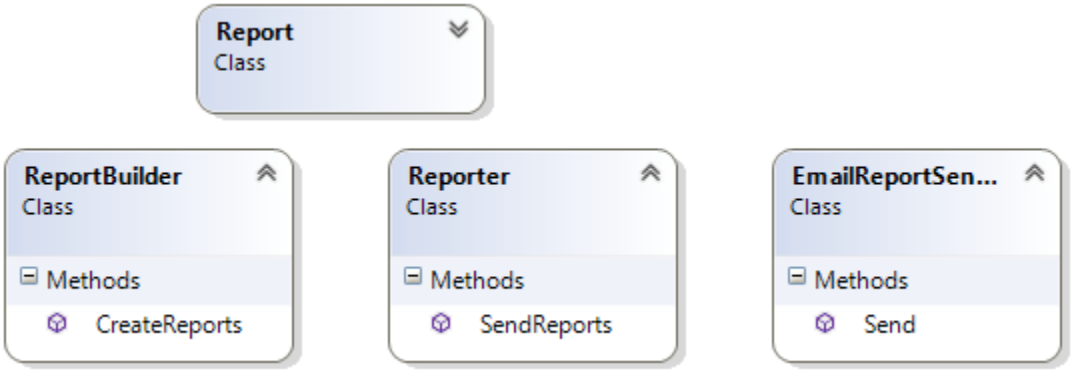
No!

- It is perfectly **safe to depend** upon concrete classes that are not going to change.
- They are not going to change (much) in the next decade, so we can feel relatively safe using them.

UML for Java (TM) Programmers by *Robert C. Martin.*

Let's practice!
What problems do you see?

The example


Report
Class
ReportBuilder
Class
Methods
CreateReports
Reporter
Class
Methods
SendReports
EmailReportSen...
Class
Methods
Send

The example

```
class Report...


class EmailReportSender
{
   public void Send(Report report)...
}

class ReportBuilder
{
   public IList<Report> CreateReports()...
}

public class Reporter
{
   public void SendReports()
   {
      var reportBuilder = new ReportBuilder();
      IList<Report> reports = reportBuilder.CreateReports();

      if (reports.Count == 0)
         throw new NoReportsException();

      var reportSender = new EmailReportSender();
      foreach (Report report in reports)
      {
         reportSender.Send(report);
      }
   }
}
```

(High) Coupling

- Knows that it will create reports **ReportBuilder**.
- Knows that all reports have to be sent via email by **EmailReportSender**.
- Can to create the object **ReportBuilder**.
- Can to create the object **EmailReportSender**.

What principles SOLID are violated?

http://siriusbuzz.com/what-happens-with-that-sirius-lifetime-subscription.php/big-question-mark

Violation of this principles

- Single responsibility principle (SOLID)
- Open/closed principle (SOLID)
- Dependency inversion principle (SOLID)

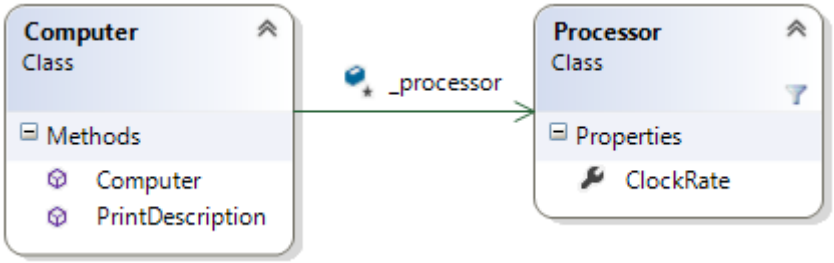
The first example

The first example

```
public class Processor
{
    private const int CLOCK_RATE = 166;
    public int ClockRate
    {
        get { return CLOCK_RATE; }
    }
}

class Computer
{
    protected Processor _processor;

    public Computer()
    {
        _processor = new Processor();
    }

    public void PrintDescription()
    {
        Console.Write("Clock rate: {0} Mhz", _processor.ClockRate);
    }
}
```

http://www.handcode.ru/2010/02/blog-post.html

The first example



How to do well?

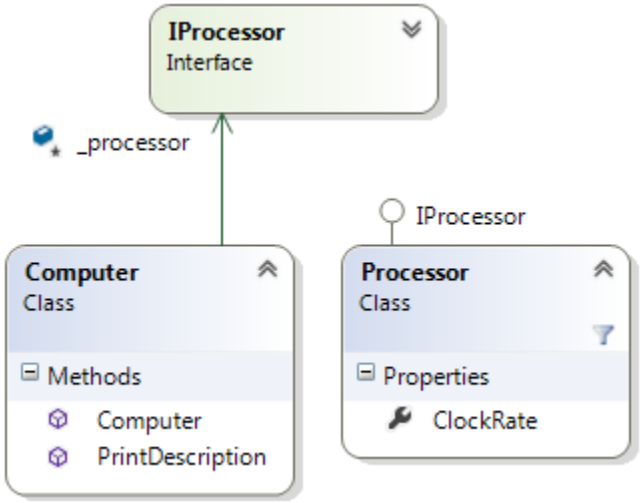
http://siriusbuzz.com/what-happens-with-that-sirius-lifetime-subscription.php/big-question-mark

The first example

```
public interface IProcessor
{
    int ClockRate { get; }
}

public class Processor : IProcessor
{
    private const int CLOCK_RATE = 166;
    public int ClockRate
    {
        get { return CLOCK_RATE; }
    }
}

public class Computer
{
    protected IProcessor _processor;

    public Computer(IProcessor processor)
    {
        _processor = processor;
    }

    public void PrintDescription()
    {
        Console.Write("Clock rate: {0} Mhz", _processor.ClockRate);
    }
}
```

http://www.handcode.ru/2010/02/blog-post.html

The first example



The second example

The second example

```
public class Product
{
   public double Cost { get; set; }
   public String Name { get; set; }
   public uint Count { get; set; }
}

public class Warehouse
{
   public IEnumerable<Product> GetProducts()
   {
      return new List<Product>
         {
          new Product {Cost = 100, Name = "Tyres", Count = 1000},
          new Product {Cost = 120, Name = "Disks", Count = 200},
          new Product {Cost = 90, Name = "Alarms", Count = 500},
          new Product {Cost = 150, Name = "Batteries", Count = 200},
          new Product {Cost = 60, Name = "Tools", Count = 50}
         };
   }
}
```

http://muradovm.blogspot.com/2012/01/blog-post.html

The second example

```
public class DiscountScheme
{
    public double GetDiscount(Product p)
    {
        if (p.Name == "Tyres")
            return 0.01;
        else if (p.Name == "Disks")
            return 0.05;
        else if (p.Name == "Alarms")
            return 0.1;
        else if (p.Name == "Batteries")
            return 0.15;
        else if (p.Name == "Tools")
            return 0.1;
        else
            return 0;
    }
}

public class ProductService
{
    public double GetAllDiscount()
    {
        double sum = 0;

        var wh = new Warehouse();
        var products = wh.GetProducts();
        var ds = new DiscountScheme();

        foreach (var p in products)
            sum += p.Cost * p.Count * ds.GetDiscount(p);

        return sum;
    }
}
```

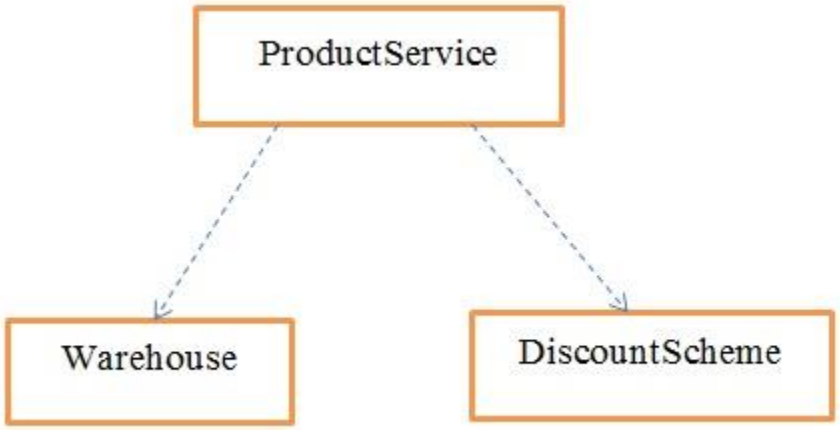
http://muradovm.blogspot.com/2012/01/blog-post.html

The second example

```
public class Program
{
   private static void Main(string[] args)
   {
      var ps = new ProductService();
      Console.WriteLine("Discount for all products = " + ps.GetAllDiscount());

      Console.ReadKey();
   }
}
```

http://muradovm.blogspot.com/2012/01/blog-post.html

The second example



http://muradovm.blogspot.com/2012/01/blog-post.html

How to do well?

http://siriusbuzz.com/what-happens-with-that-sirius-lifetime-subscription.php/big-question-mark

The second example

```
public interface IProductStorage
{
   IEnumerable<Product> GetProducts();
}

public interface IDiscountCalculator
{
   double GetDiscount(Product products);
}

public class Product
{
   public double Cost { get; set; }
   public String Name { get; set; }
   public uint Count { get; set; }
}

public class Warehouse : IProductStorage
{
   public IEnumerable<Product> GetProducts()
   {
      return new List<Product>
         {
           new Product {Cost = 100, Name = "Tyres", Count = 1000},
           new Product {Cost = 120, Name = "Disks", Count = 200},
           new Product {Cost = 90, Name = "Alarms", Count = 500},
           new Product {Cost = 150, Name = "Batteries", Count = 200},
           new Product {Cost = 60, Name = "Tools", Count = 50}
         };
   }
}
```

http://muradovm.blogspot.com/2012/01/blog-post.html

The second example

```
public class SimpleScheme : IDiscountCalculator
{
    public double GetDiscount(Product p)
    {
        if (p.Name == "Tyres")
            return 0.01;
        else if (p.Name == "Disks")
            return 0.05;
        else if (p.Name == "Alarms")
            return 0.1;
        else if (p.Name == "Batteries")
            return 0.15;
        else if (p.Name == "Tools")
            return 0.1;
        else
            return 0;
    }
}

public class ProductService
{
    public double GetAllDiscount(IProductStorage storage,
                    IDiscountCalculator discountCalculator)
    {
        double sum = 0;

        foreach (var p in storage.GetProducts())
            sum += p.Cost * p.Count * discountCalculator.GetDiscount(p);

        return sum;
    }
}
```

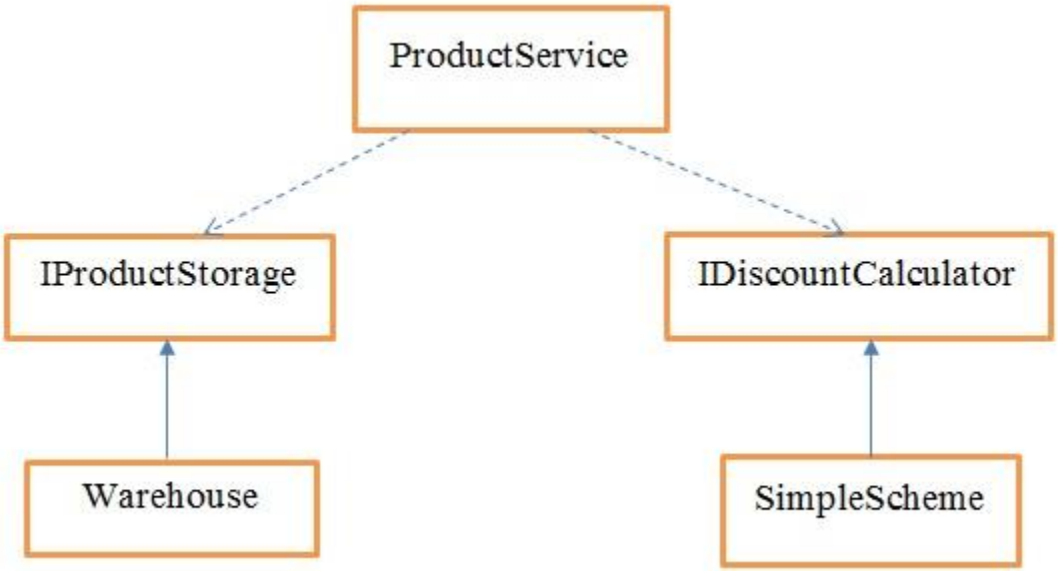
http://muradovm.blogspot.com/2012/01/blog-post.html

The second example

```
class Program
{
   static void Main(string[] args)
   {
      ProductService ps = new ProductService();
      Console.WriteLine("Discount for all products = " +
                ps.GetAllDiscount(new Warehouse(), new SimpleScheme()));

      Console.ReadKey();
   }
}
```

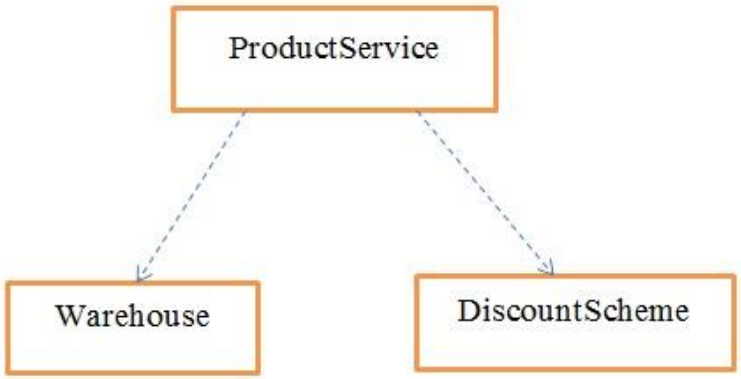
http://muradovm.blogspot.com/2012/01/blog-post.html

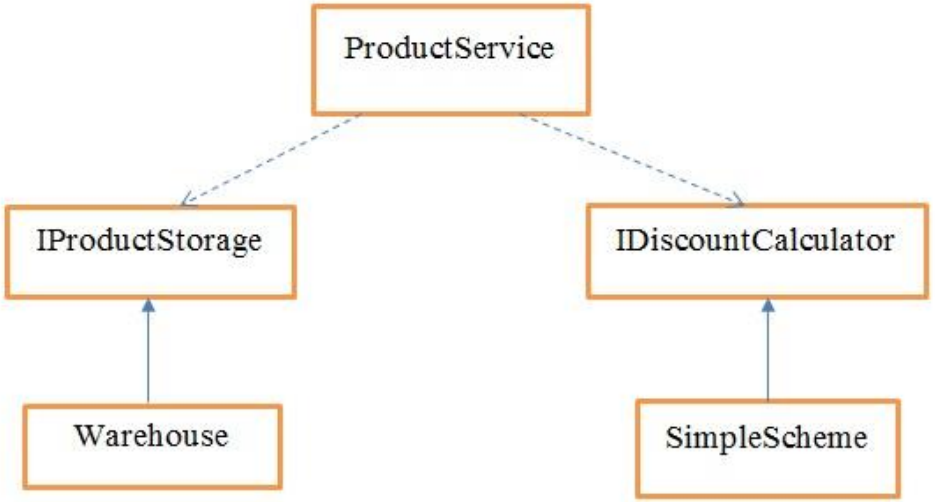
The second example



http://muradovm.blogspot.com/2012/01/blog-post.html

Dependency Inversion Principle





http://muradovm.blogspot.com/2012/01/blog-post.html

The third example

The third example

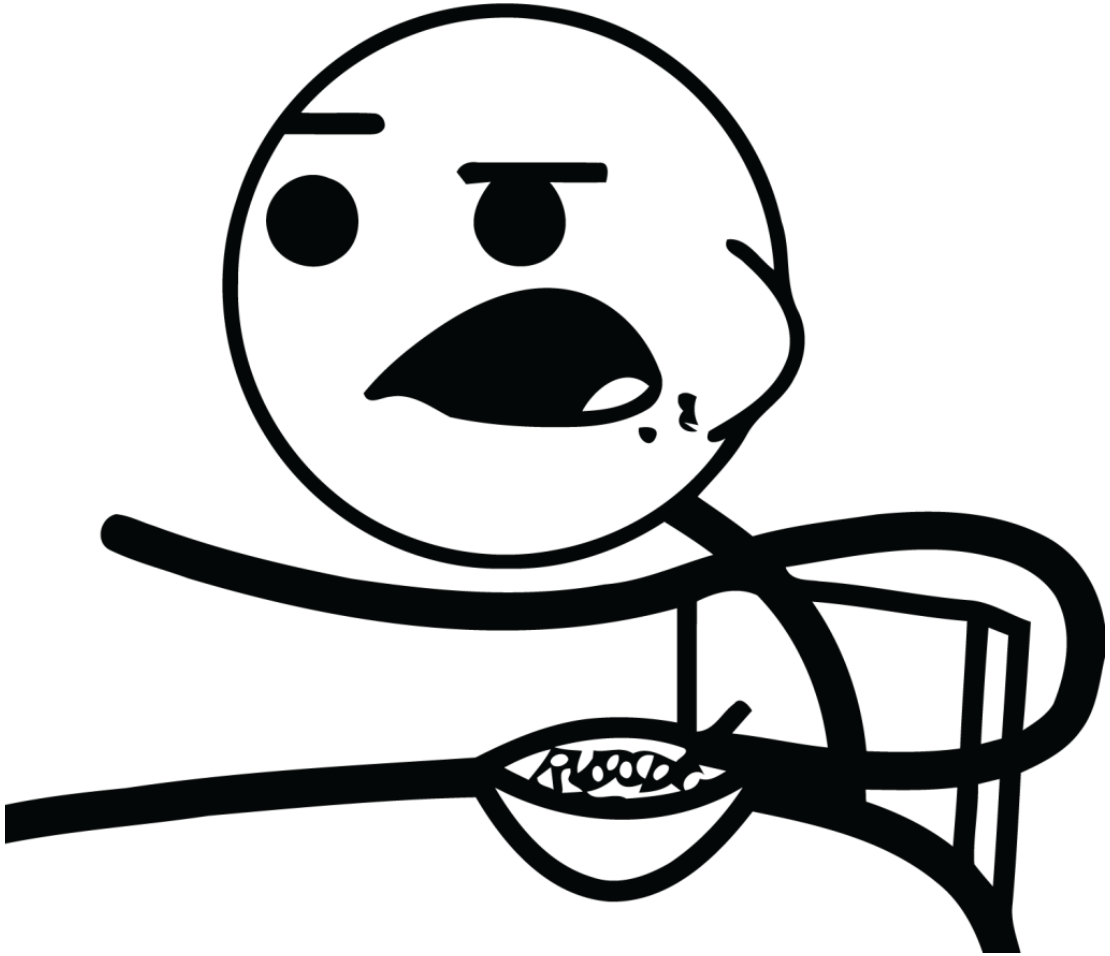
```
public class EncryptionService
{
   public void Encrypt(string sourceFileName, string targetFileName)
   {
      // read the contents of the file
      byte[] content;
      using (var fs = new FileStream(sourceFileName, FileMode.Open, FileAccess.Read))
      {
         content = new byte[fs.Length];
         fs.Read(content, 0, content.Length);
      }

      // do encryption
      byte[] encryptedContent = DoEncryption(content);

      // write encrypted data
      using (var fs = new FileStream(targetFileName, FileMode.CreateNew, FileAccess.ReadWrite))
      {
         fs.Write(encryptedContent, 0, encryptedContent.Length);
      }
   }

   private byte[] DoEncryption(byte[] content)
   {
      byte[] encryptedContent = null;
      // here is the encryption algorithm ..
      return encryptedContent;
   }
}
```

http://silverlight.su/viewtopic.php?id=166

The requirements have changed!

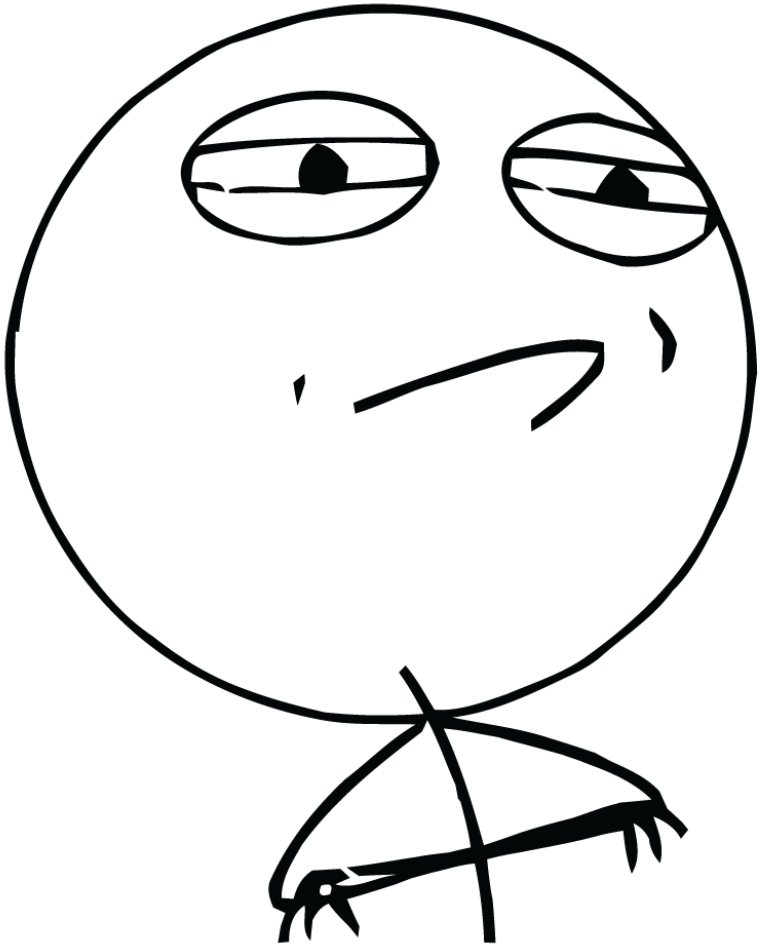
It is also possible to download data from the database and write using the web services.

Again, change the requirements?



. . .

CHALLENGE ACCEPTED

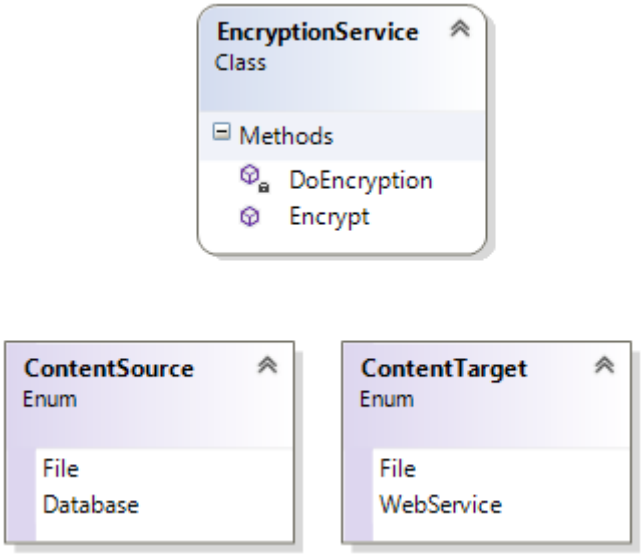


The third example

```
public class EncryptionService
{
    public void Encrypt(ContentSource source, ContentTarget target)
    {
        // read data
        byte[] content;
        switch (source)
        {
            case ContentSource.File: content = GetFromFile(); break;
            case ContentSource.Database: content = GetFromDatabase(); break;
        }

        // do encryption
        byte[] encryptedContent = DoEncryption(content);

        // write encrypted data
        switch (target)
        {
            case ContentTarget.File: WriteToFile(encryptedContent); break;
            case ContentTarget.WebService: WriteToWebService(encryptedContent); break;
        }
    }

    // rest of the code omitted for brevity
    private void WriteToWebService(byte[] encryptedContent)...
    private void WriteToFile(byte[] encryptedContent)...
    private byte[] GetFromDatabase()...
    private byte[] GetFromFile()...
    private byte[] DoEncryption(byte[] content)...
}
```

http://silverlight.su/viewtopic.php?id=166

The third example



How to do well?

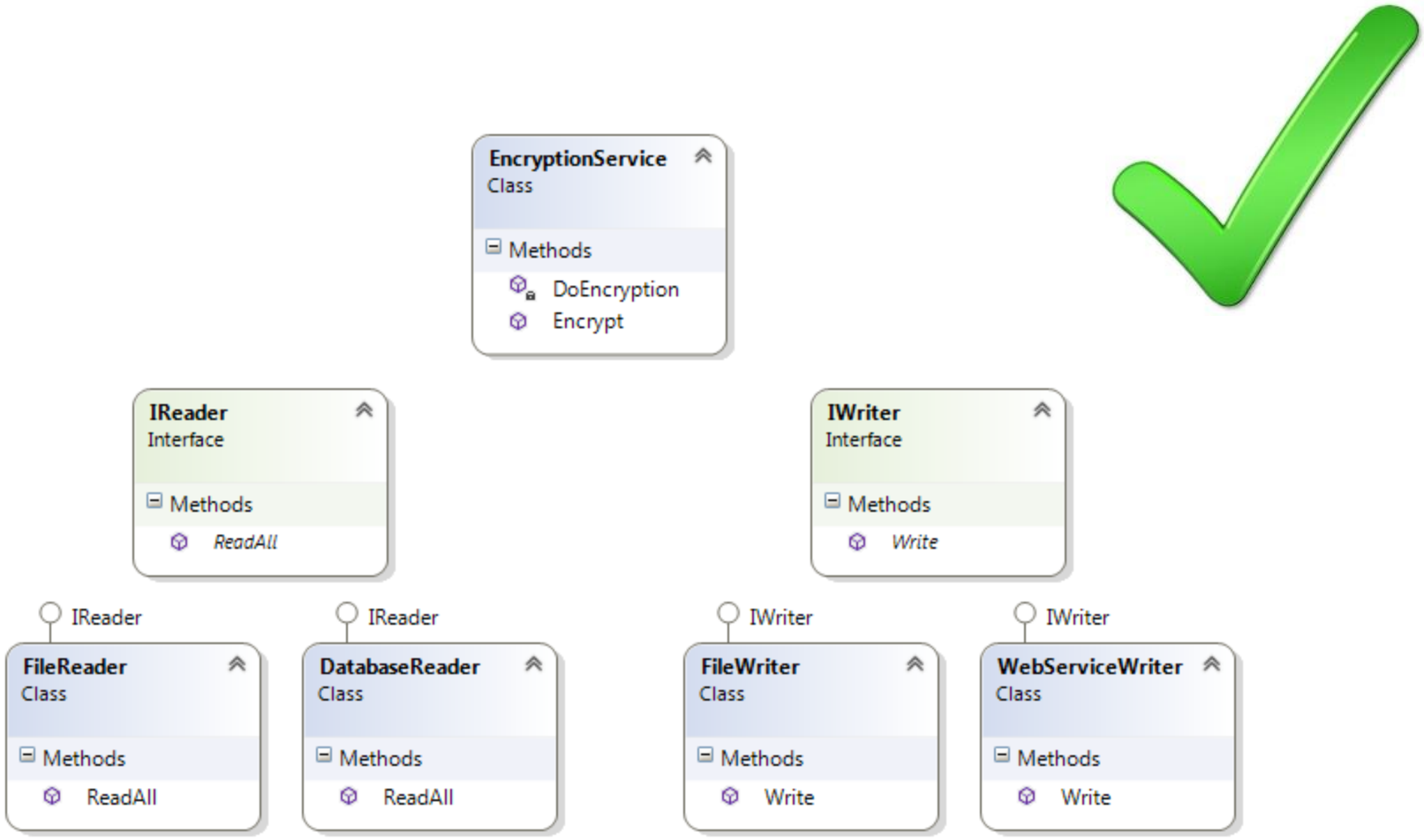
http://siriusbuzz.com/what-happens-with-that-sirius-lifetime-subscription.php/big-question-mark

The third example

```
public interface IReader
{
    byte[] ReadAll();
}

public interface IWriter
{
    void Write(byte[] encryptedContent);
}

public class EncryptionService
{
    public void Encrypt(IReader reader, IWriter writer)
    {
        // read data
        byte[] content = reader.ReadAll();

        // do encryption
        byte[] encryptedContent = DoEncryption(content);

        // write encrypted data
        writer.Write(encryptedContent);
    }

    // rest of code
}
```

http://silverlight.su/viewtopic.php?id=166

The third example



In summary

- The DIP makes clients reusable by abstracting the interface the client needs from a server from the server's implementation.
- This protects the client's design from depending on incidental (as opposed to fundamental) aspects of its server.
- Thus the DIP is good practice even when the client is not intended to be reused.

http://www.engr.mun.ca/~theo/Courses/sd/5895-downloads/sd-principles-4.ppt.pdf

Resources

Books and papers

- **The Dependency Inversion Principle** by *Robert C. Martin.*
- **Object-Oriented Analysis and Design with Applications** by *Grady Booch*.
- **UML for Java (TM) Programmers** by *Robert C. Martin.*
- **Head First Design Patterns** by *Elisabeth Freeman, Eric Freeman, Bert Bates and Kathy Sierra.*
- *…*