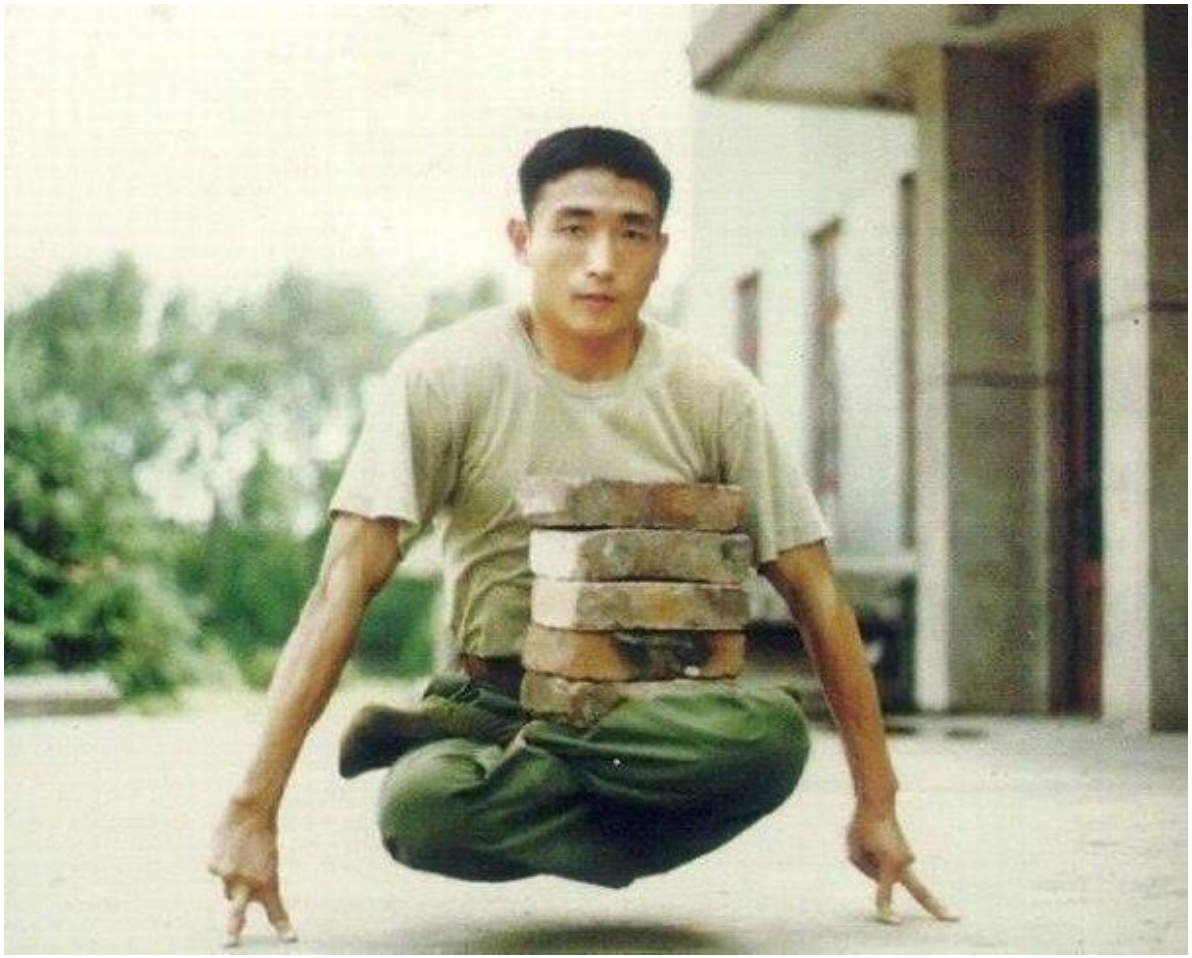
# Good coding practice in real life

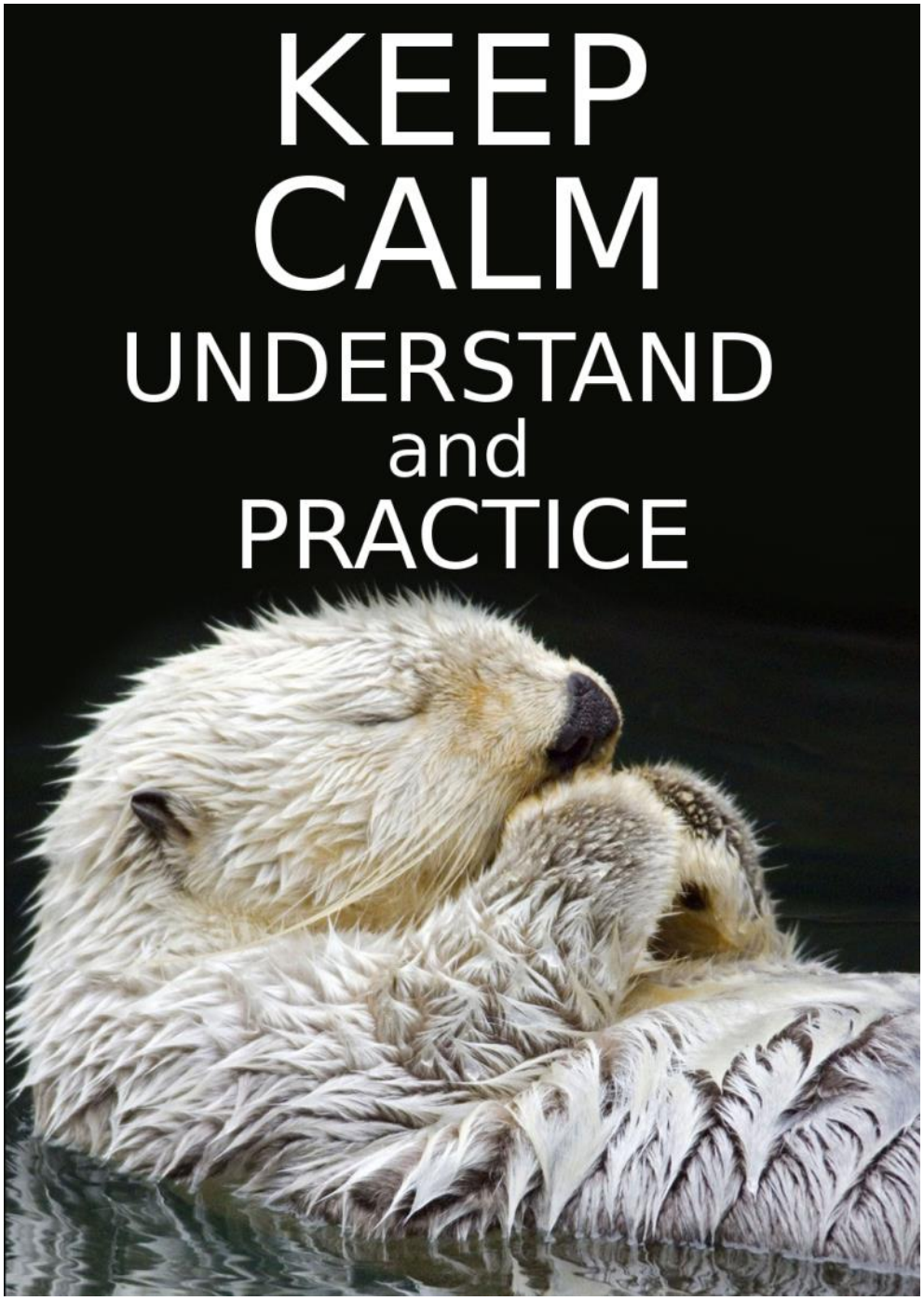
*(with a focus on ISP)*

Vladimir Alekseichenko

# Good practices make life easier

# Good practices are not easy




KEEP
CALM
UNDERSTAND
and
PRACTICE

# Agenda

## The interface segregation principle

- The definition ISP.
- A fat interface.
- The example from **.Net**.
- Some examples violating ISP and solution for them.

# Robert Cecil Martin aka "Uncle Bob"

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

# Interface segregation principle

**Clients** should not be forced to **depend** upon interfaces that they do not use.

This principle deals with the disadvantages of „fat” interfaces.

www.objectmentor.com/resources/articles/isp.pdf

# What does mean a fat interface?

# I’m not fat, I’m big boned

# A fat interface

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.

www.objectmentor.com/resources/articles/isp.pdf

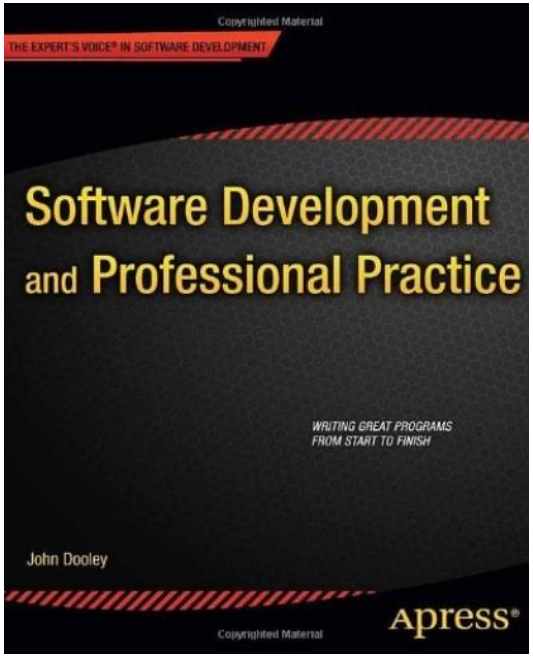
# Low cohesion

http://persbaglio.ilcannocchiale.it/?TAG=disordine

# High cohesion

http://www.3deko.info/inter/ofor/415-uyutnyj-balkon-mesto-otdyxa-chast-ii.html

# John Dooley




http://www.knox.edu/news-and-events/news-archive/faculty-lecture-on-turing-and-enigma-machine.html

# Interface segregation principle

In particular, they shouldn't have to **depend** on methods they **don't use**.

# A greatest temptations

http://www.taday.ru/text/204597.html

# A greatest temptations

One of the greatest temptations with respect to **interfaces** is to **make them bigger**.

# A greatest temptations

If an interface is good, then a bigger interface **must be better, right?**

# …a bigger interface **must be better, right?**

# Interface segregation principle

You can then use the interface is way more objects and the user just has to not implement certain methods that they don't need.

# Interface segregation principle

## Remember (high) **cohesion** is **good**.

Your applications should be cohesive and the classes and interfaces they depend on should also be cohesive.

# Violate the ISP

You make your interfaces less cohesive, and begin to **violate** the **ISP.**

When you start adding new methods to your interface because one of the subclasses that implements the interface needs it – and others do not.

# Interface cohesive

How do we **keep our interfaces cohesive** and still make them **useful for a range of objects**?

# So what's the answer here?

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

# Interface cohesive

# Make more interfaces.

# Interface cohesive

# You divide your bloated interface into two or more smaller, more cohesive interfaces.

The ISP implies that instead of adding new methods that are only appropriate to one or a few implementation classes, that you make a new interface.

# Interface cohesive

That way, new classes can just **implement** the interfaces that they **need** and **not implement** ones that they **don't**.

# The example from .Net

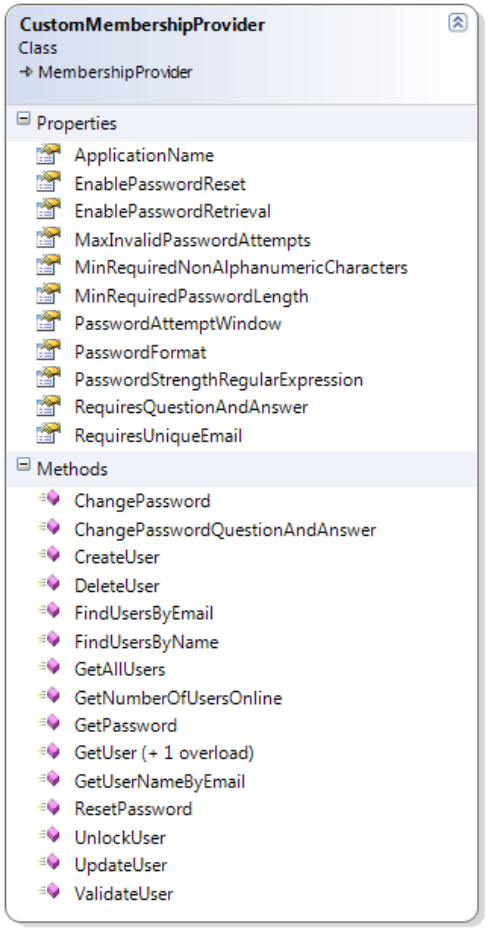
# MembershipProvider

```
public class CustomMembershipProvider : MembershipProvider
{
    public override string ApplicationName
    {
        get
        {
            throw new Exception("The method or operation is not implemented.");
        }
        set
        {
            throw new Exception("The method or operation is not implemented.");
        }
    }

    public override bool ChangePassword(string username, string oldPassword, string newPassword
    {
        throw new Exception("The method or operation is not implemented.");
    }

    public override bool ChangePasswordQuestionAndAnswer(string username, string password,
        string newPasswordQuestion, string newPasswordAnswer)
    {
        throw new Exception("The method or operation is not implemented.");
```

# MembershipProvider

```
{
    get { throw new Exception("The method or operation is not implemented."); }
}

public override string ResetPassword(string username, string answer)
{
    throw new Exception("The method or operation is not implemented.");
}

public override bool UnlockUser(string userName)
{
    throw new Exception("The method or operation is not implemented.");
}

public override void UpdateUser(MembershipUser user)
{
    throw new Exception("The method or operation is not implemented.");
}

public override bool ValidateUser(string username, string password)
{
    throw new Exception("The method or operation is not implemented.");
}
```

# MembershipProvider




← 11!

← 15!

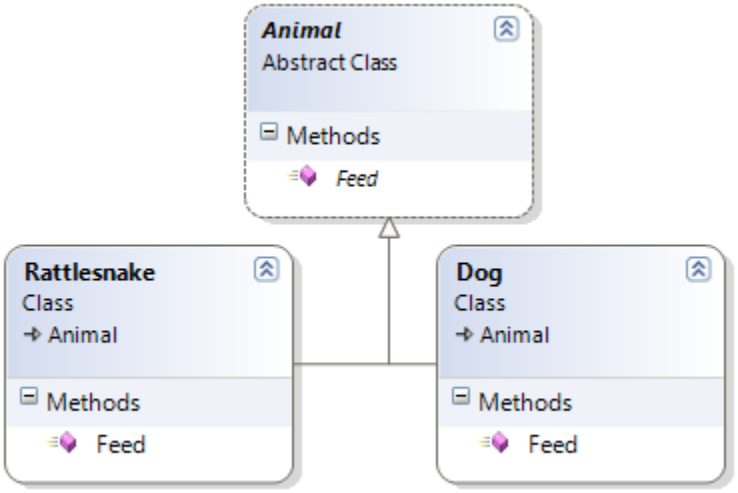
# The first example

# The first example

```
public abstract class Animal
{
   public abstract void Feed();
}

public class Dog : Animal
{
   public override void Feed()
   {
      // do something
   }
}

public class Rattlesnake : Animal
{
   public override void Feed()
   {
      // do something
   }
}
```

http://lostechies.com/rayhouston/2008/03/15/ptom-the-interface-segregation-principle

# The first example

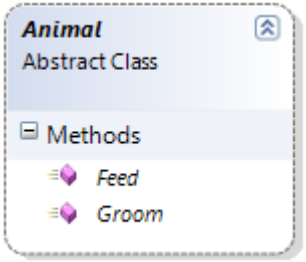


# The first example

But then you realize that you have a need for some of the animals to be treated as pets and have them groomed.

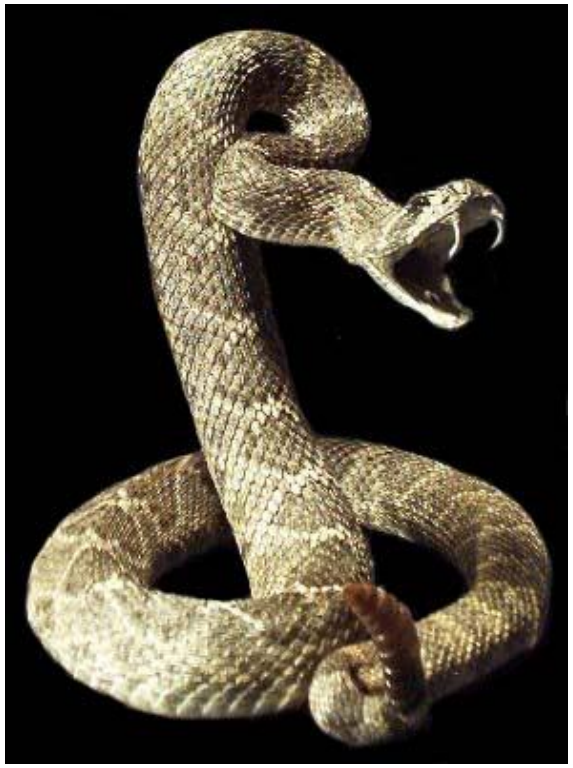
You may be tempted to do:

```
public abstract class Animal
{
   public abstract void Feed();
   public abstract void Groom();
}
```




http://lostechies.com/rayhouston/2008/03/15/ptom-the-interface-segregation-principle

# Do you prefer to groom of a **dog** or a **rattlesnake**?

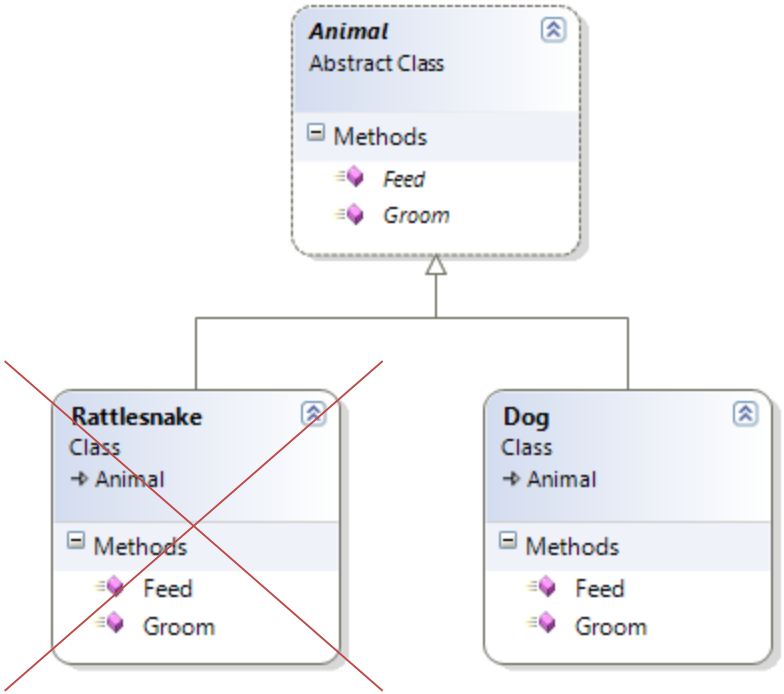


# The first example

```
public abstract class Animal
{
    public abstract void Feed();
    public abstract void Groom();
}

public class Dog : Animal
{
    public override void Feed()
    {
        //do something
    }

    public override void Groom()
    {
        //do something
    }
}

public class Rattlesnake : Animal
{
    public override void Feed()
    {
        // do something
    }

    public override void Groom()
    {
        // ignore - I'm not grooming a freaking rattlesnake
    }
}
```

http://lostechies.com/rayhouston/2008/03/15/ptom-the-interface-segregation-principle

# The first example




Here we have **violated the ISP** by polluting our **Animal** interface.

http://lostechies.com/rayhouston/2008/03/15/ptom-the-interface-segregation-principle

# How to do well?

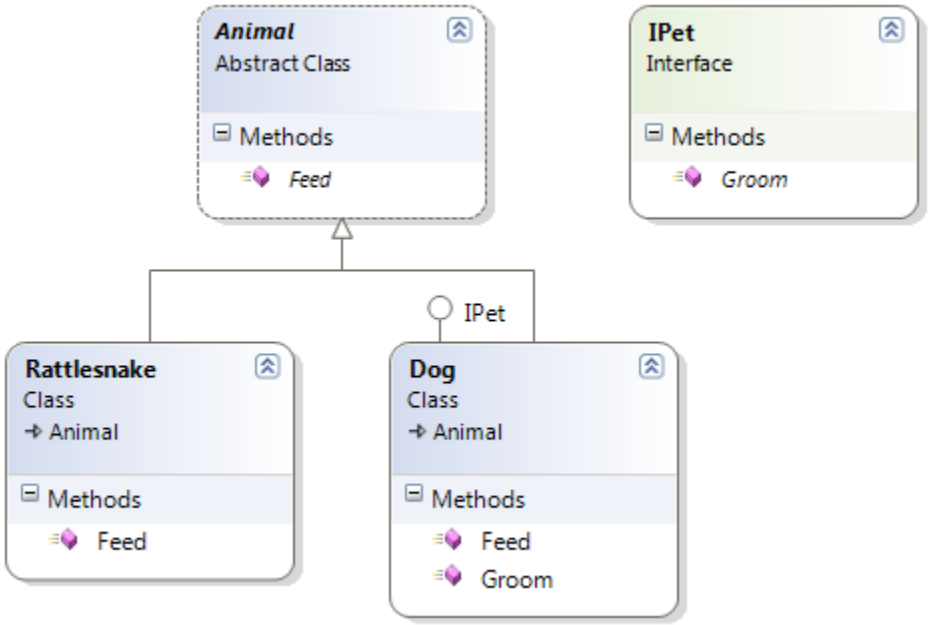
http://siriusbuzz.com/what-happens-with-that-sirius-lifetime-subscription.php/big-question-mark

# The solution

```
public abstract class Animal
{
   public abstract void Feed();
}

public interface IPet
{
   void Groom();
}

public class Dog : Animal, IPet
{
   public override void Feed()
   {
      // do something
   }

   public void Groom()
   {
      // do something
   }
}

public class Rattlesnake : Animal
{
   public override void Feed()
   {
      // do something
   }
}
```

http://lostechies.com/rayhouston/2008/03/15/ptom-the-interface-segregation-principle

# The solution



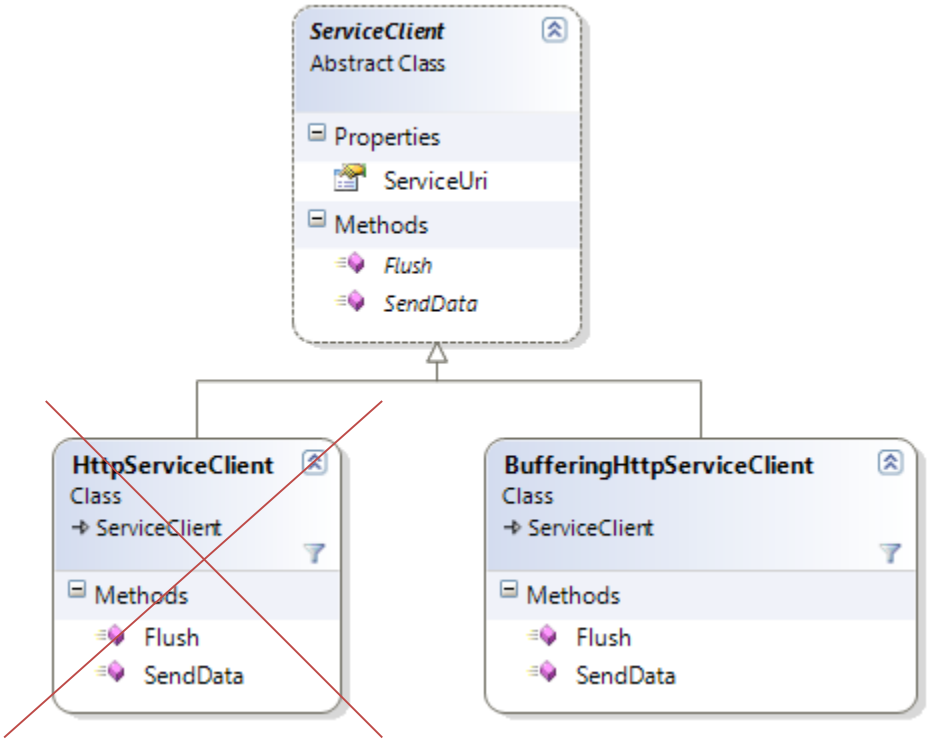

# The second example

# The second example

```
abstract class ServiceClient
{
   public string ServiceUri { get; set; }
   public abstract void SendData(object data);
   public abstract void Flush();
}

class HttpServiceClient : ServiceClient
{
   public override void SendData(object data)
   {
      var channel = OpenChannel(ServiceUri);
      channel.Send(data);
   }

   public override void Flush()
   {
      //to do nothing
   }


}

class BufferingHttpServiceClient : ServiceClient
{
   public override void SendData(object data)
   {
      Buffer.Write(data);
   }

   public override void Flush()
   {
      var channel = OpenChannel(ServiceUri);
      channel.Send(Buffer.GetAll());
   }


}
```

http://igor.quatrocode.com/2008/09/solid-top-5.html

# The second example




Here we have **violated the ISP** by polluting our **ServiceClient** interface.

http://igor.quatrocode.com/2008/09/solid-top-5.html

# How to do well?

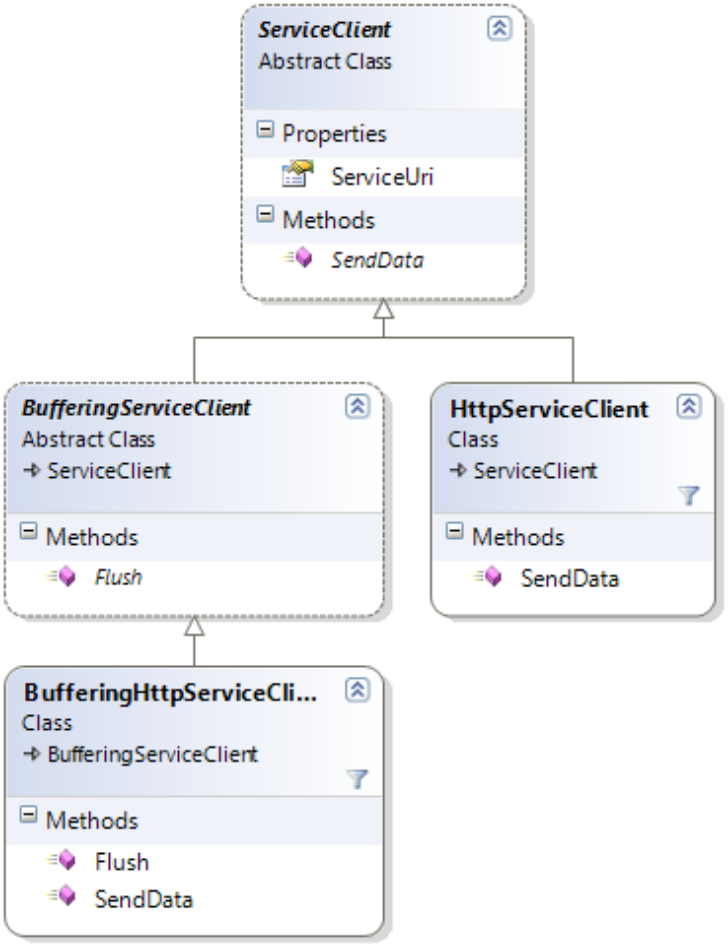
http://siriusbuzz.com/what-happens-with-that-sirius-lifetime-subscription.php/big-question-mark

# The solution

```
abstract class ServiceClient
{
   public string ServiceUri { get; set; }
   public abstract void SendData(object data);
}

abstract class BufferingServiceClient : ServiceClient
{
   public abstract void Flush();
}

class HttpServiceClient : ServiceClient
{
   public override void SendData(object data)
   {
      var channel = OpenChannel(ServiceUri);
      channel.Send(data);
   }
   private Channel OpenChannel(string serviceUri)...
}

class BufferingHttpServiceClient : BufferingServiceClient
{
   public override void SendData(object data)
   {
      Buffer.Write(data);
   }

   public override void Flush()
   {
      var channel = OpenChannel(ServiceUri);
      channel.Send(Buffer.GetAll());
   }
   private Channel OpenChannel(string serviceUri)...
}
```

http://igor.quatrocode.com/2008/09/solid-top-5.html

# The solution



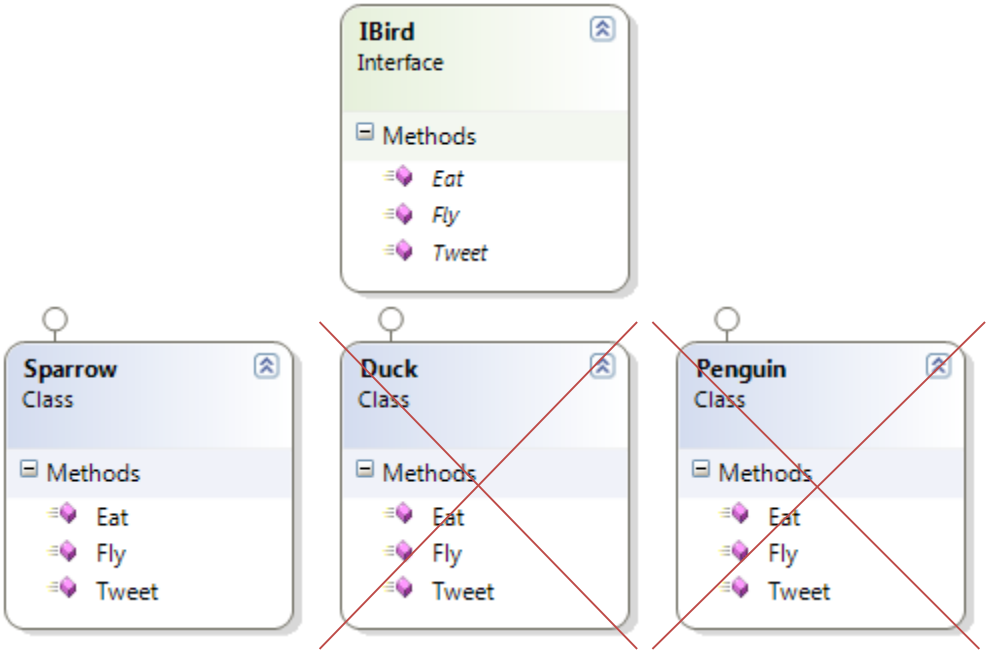

# The third example

# The third example

```
interface IBird
{
   void Fly();
   void Eat();
   void Tweet();
}

class Sparrow : IBird
{
   public void Fly() { /*...*/ }
   public void Eat() { /*...*/ }
   public void Tweet() { /*...*/ }
}

class Duck : IBird
{
   public void Fly() { /*...*/ }
   public void Eat() { /*...*/ }
   public void Tweet() { throw new NotSupportedException(); }
}

class Penguin : IBird
{
   public void Fly() { throw new NotSupportedException(); }
   public void Eat() { /*...*/ }
   public void Tweet() { throw new NotSupportedException(); }
}
```

http://dimakudr.blogspot.com/2010/07/solid.html

# The third example




Here we have **violated the ISP** by polluting our **IBird** interface.

# How to do well?

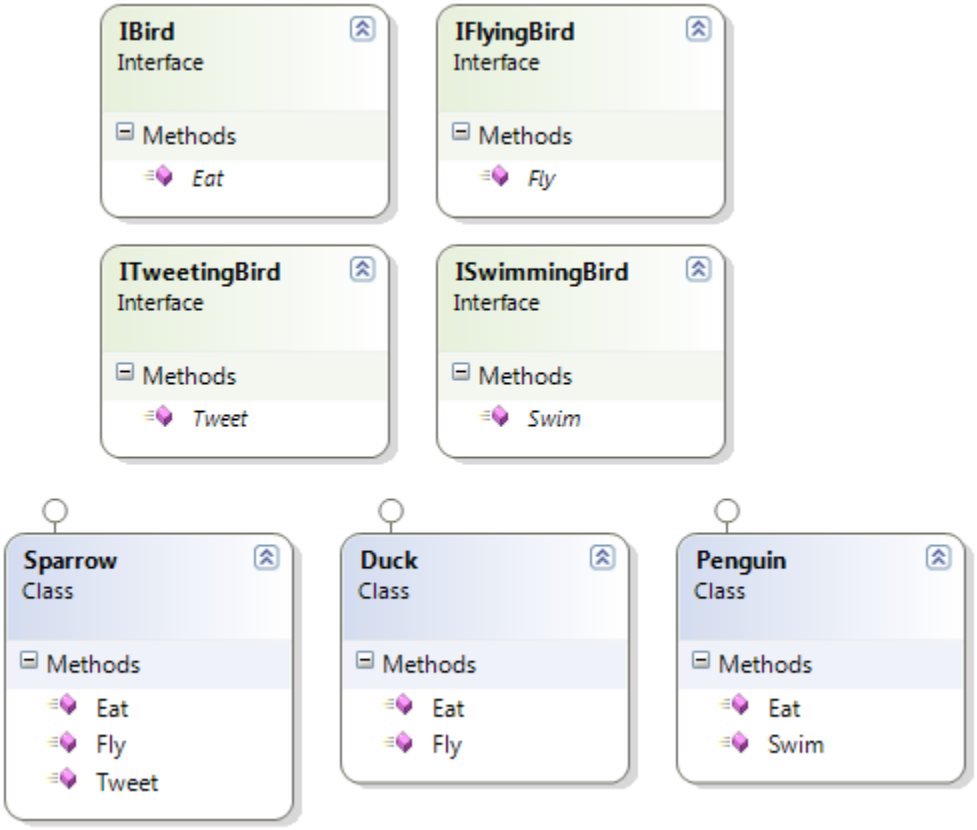
http://siriusbuzz.com/what-happens-with-that-sirius-lifetime-subscription.php/big-question-mark

# The solution

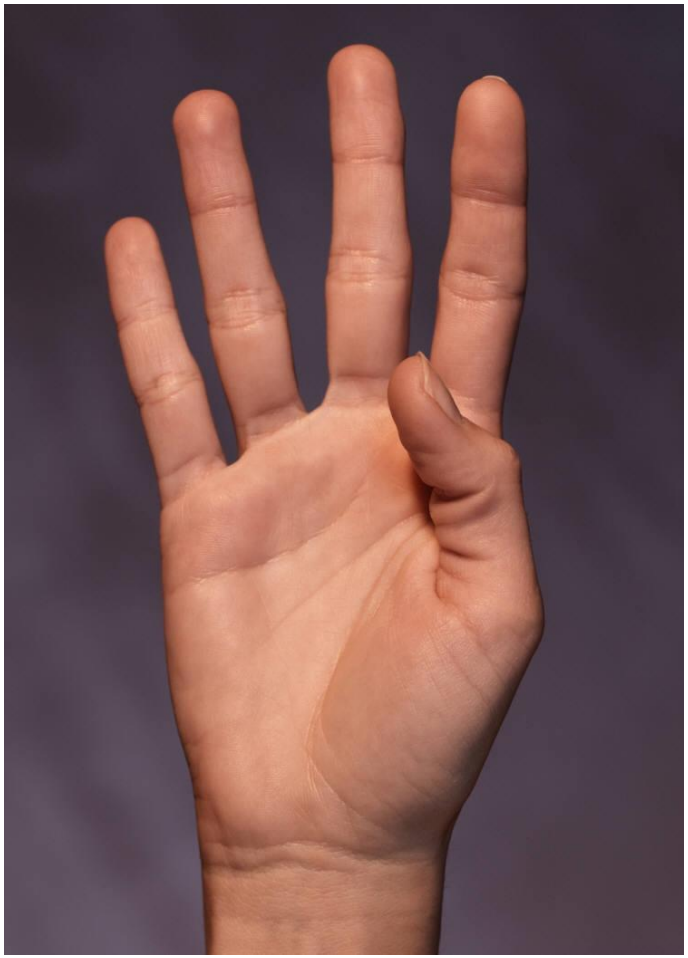
```
interface IBird
{
   void Eat();
}

interface IFlyingBird
{
   void Fly();
}

interface ITweetingBird
{
   void Tweet();
}

interface ISwimmingBird
{
   void Swim();
}

class Sparrow : IBird, IFlyingBird, ITweetingBird
{
   public void Fly() { /*...*/ }
   public void Eat() { /*...*/ }
   public void Tweet() { /*...*/ }
}

class Duck : IBird, IFlyingBird
{
   public void Fly() { /*...*/ }
   public void Eat() { /*...*/ }
}

class Penguin : IBird, ISwimmingBird
{
   public void Eat() { /*...*/ }
   public void Swim() { /*...*/ }
}
```

http://dimakudr.blogspot.com/2010/07/solid.html

# The solution




# The fourth example

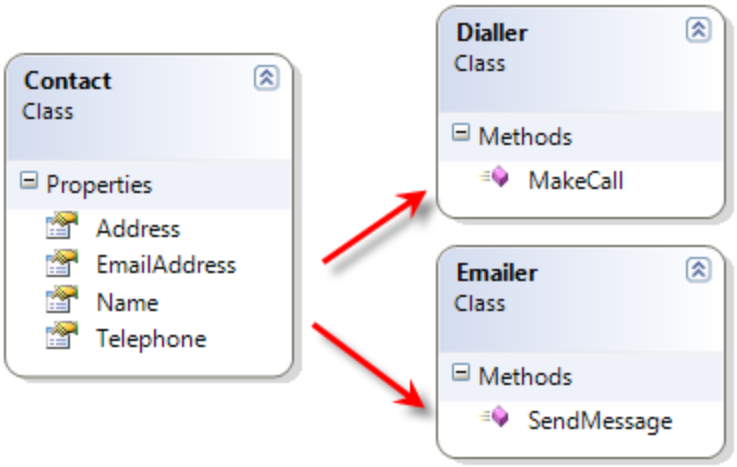


# The fourth example

```
public class Contact
{
   public string Name { get; set; }
   public string Address { get; set; }
   public string EmailAddress { get; set; }
   public string Telephone { get; set; }
}

public class Emailer
{
   public void SendMessage(Contact contact, string subject, string body)
   {
      // Code to send email, using contact's email address and name
   }
}

public class Dialler
{
   public void MakeCall(Contact contact)
   {
      // Code to dial telephone number of contact
   }
}
```

http://www.blackwasp.co.uk/ISP.aspx

# The fourth example




- The Contact class represents a person or business that can be contacted.
- The example code violates the ISP.

http://www.blackwasp.co.uk/ISP.aspx

# How to do well?

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

# The solution

```
public interface IEmailable
{
   string Name { get; set; }
   string EmailAddress { get; set; }
}

public interface IDiallable
{
   string Telephone { get; set; }
}

public class Contact : IEmailable, IDiallable
{
   public string Name { get; set; }
   public string Address { get; set; }
   public string EmailAddress { get; set; }
   public string Telephone { get; set; }
}

public class MobileEngineer : IDiallable
{
   public string Name { get; set; }
   public string Vehicle { get; set; }
   public string Telephone { get; set; }
}
```

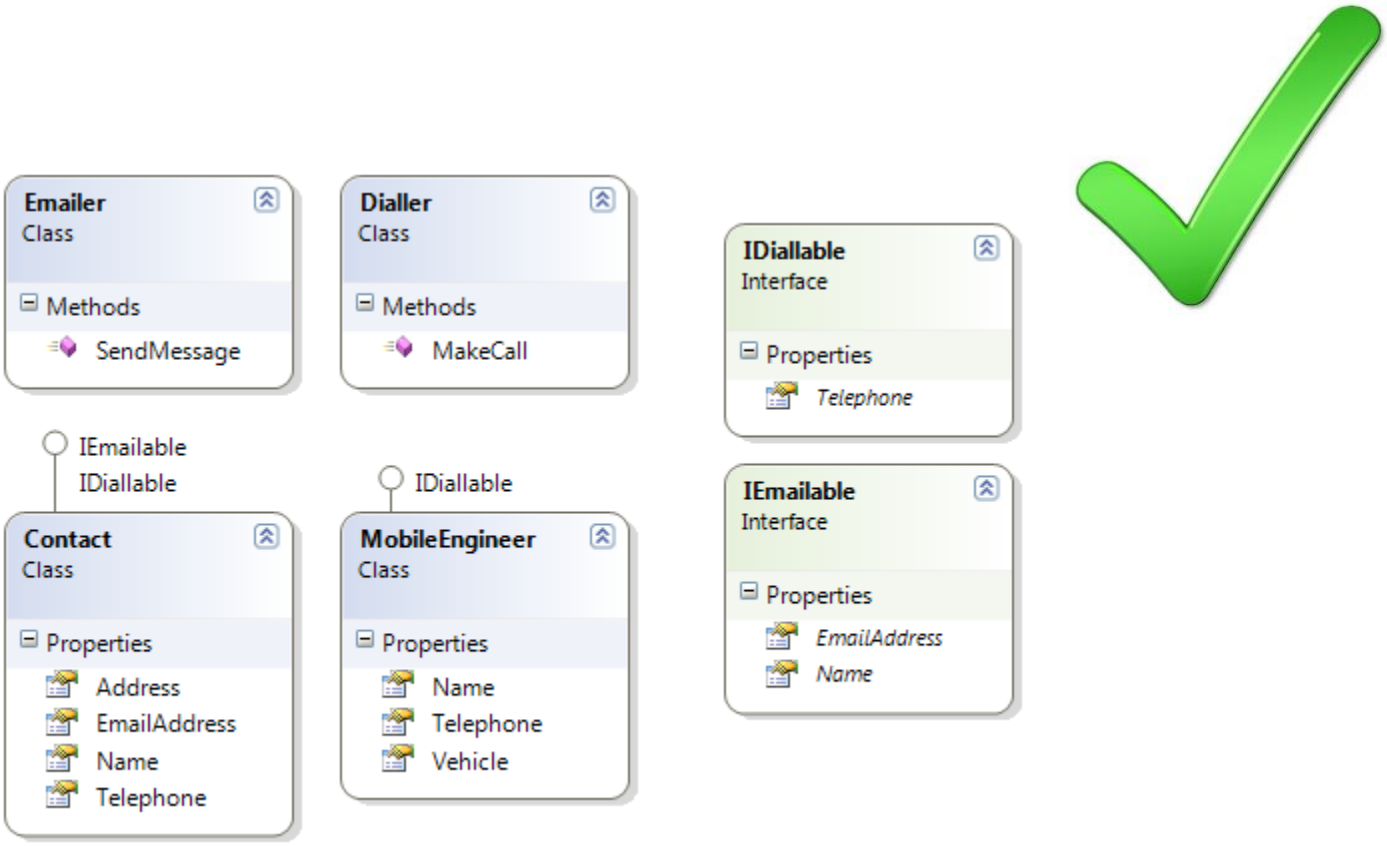
http://www.blackwasp.co.uk/ISP.aspx

# The solution

```
public class Emailer
{
    public void SendMessage(IEmailable target, string subject, string body)
    {
        // Code to send email, using target's email address and name
    }
}

public class Dialler
{
    public void MakeCall(IDiallable target)
    {
        // Code to dial telephone number of target
    }
}
```

http://www.blackwasp.co.uk/ISP.aspx

# The solution



# In summary

- Fat classes cause bizarre couplings between their clients.
- Avoid „fat” interfaces or interface pollution
- Many specific interfaces are better than a single, general interface.
- There are objects that need to be fat, that need to be noncohesive.
- Remember (high) **cohesion** is **good**.

# Resources

- **The Interface Segregation Principle** by *Robert C. Martin.*
- **Software Development and Professional Practice** by *John Dooley.*
- **Java Design: Objects, UML, and Process** by *Kirk Knoernschild.*
- **Design Principles and Patterns** by *Robert C. Martin.*
- **UML for Java (TM) Programmers** by *Robert C. Martin.*
- **Agile Software Development, Principles, Patterns, and Practices** by *Robert C. Martin.*