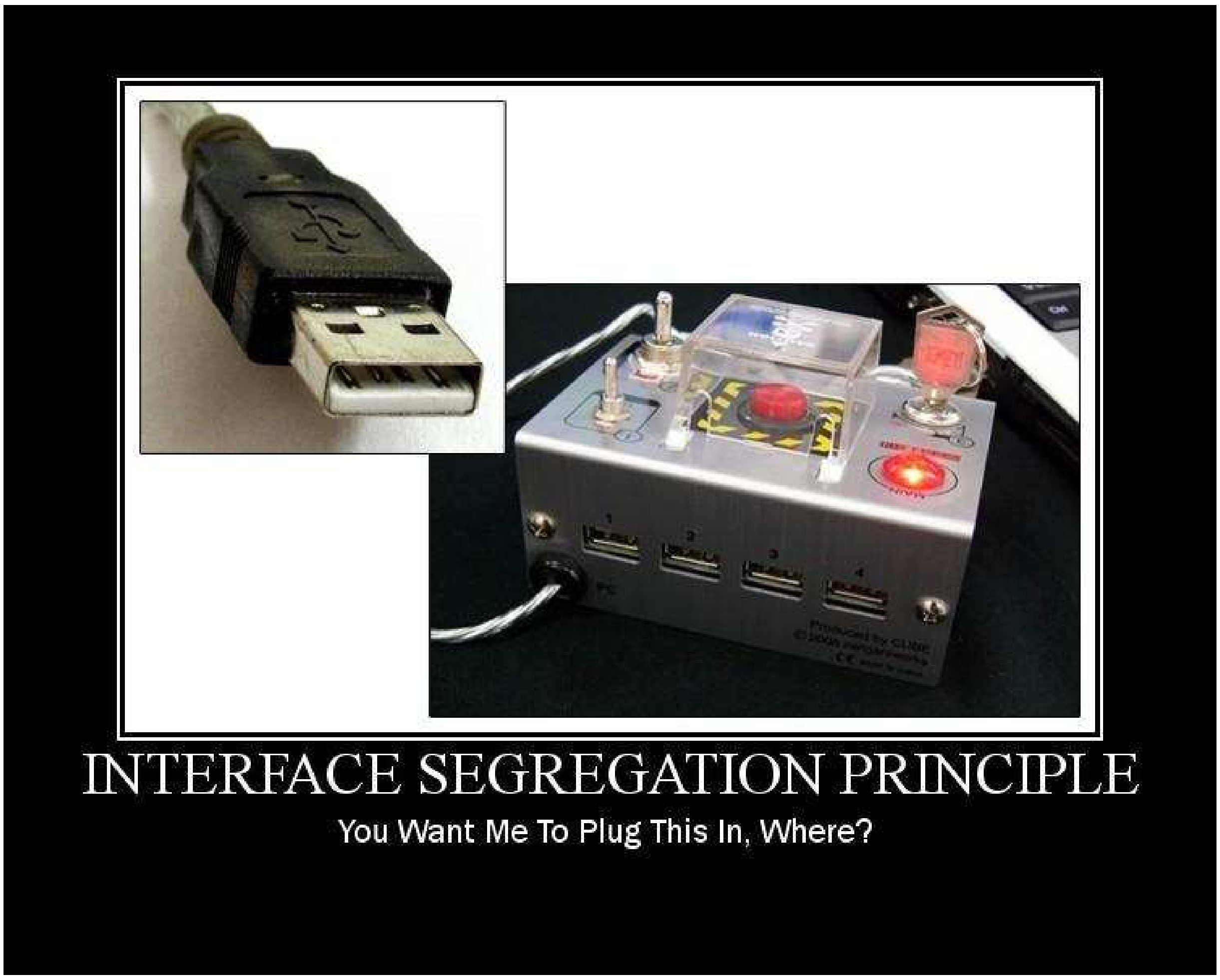
ISP: Interface Segregation Principle

CLIENTS SHOULD NOT BE FORCED TO DEPEND UPON INTERFACES THAT THEY DO NOT USE
http://www.objectmentor.com/resources/articles/isp.pdf


INTERFACE SEGREGATION PRINCIPLE
You Want Me To Plug This In, Where?

Interface Segregation Principle by Ray Houston

In following suite with the The Los Techies Pablo's Topic of the Month - March: SOLID Principles, I chose to write a little about the The Interface Segregation Principle (ISP). As Chad pointed out with LSP, the ISP is also one of Robert 'Uncle Bob' Martin's S.O.L.I.D design principles.

Basically ISP tells us that clients shouldn't be forced to implement interfaces they don't use. In other words, if you have an abstract class or an interface, then the implementers should not be forced to implement parts that they don't care about.

I was having trouble thinking of a real world example for ISP but then was reminded about implementing a custom Membership Provider in ASP.NET 2.0. I had completely blocked that monstrosity out of my mind (for good reason).

The Membership Provider was a way to integrate with some of the ASP.NET's built in management of users and its associated server controls. For me, it ended up being a lot more trouble than it was worth, but it turns out to be a good example of a fat interface. In order to implement your own Membership Provider you "simply" implement the abstract class MembershipProvider like so:

```
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.");
  }
```

```
public override MembershipUser CreateUser(string username, string password, string email,
   string passwordQuestion, string passwordAnswer, bool isApproved, object providerUserKey,
   out MembershipCreateStatus status)
{
   throw new Exception("The method or operation is not implemented.");
}

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

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

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

public override MembershipUserCollection FindUsersByEmail(string emailToMatch, int pageIndex,
   int pageSize, out int totalRecords)
{
   throw new Exception("The method or operation is not implemented.");
}

public override MembershipUserCollection FindUsersByName(string usernameToMatch, int pageIndex,
   int pageSize, out int totalRecords)
{
   throw new Exception("The method or operation is not implemented.");
}

public override MembershipUserCollection GetAllUsers(int pageIndex, int pageSize, out int totalRecords)
{
   throw new Exception("The method or operation is not implemented.");
}

public override int GetNumberOfUsersOnline()
{
   throw new Exception("The method or operation is not implemented.");
}

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

public override MembershipUser GetUser(string username, bool userIsOnline)
{
   throw new Exception("The method or operation is not implemented.");
}

public override MembershipUser GetUser(object providerUserKey, bool userIsOnline)
{
   throw new Exception("The method or operation is not implemented.");
}

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

public override int MaxInvalidPasswordAttempts
{
   get { throw new Exception("The method or operation is not implemented."); }
}

public override int MinRequiredNonAlphanumericCharacters
{
   get { throw new Exception("The method or operation is not implemented."); }
}

public override int MinRequiredPasswordLength
{
   get { throw new Exception("The method or operation is not implemented."); }
}

public override int PasswordAttemptWindow
{
   get { throw new Exception("The method or operation is not implemented."); }
}

public override MembershipPasswordFormat PasswordFormat
{
   get { throw new Exception("The method or operation is not implemented."); }
}

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

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

public override bool RequiresUniqueEmail
{
   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.");
  }
}
```

Holy guacamole! That's a lot of stuff. Sorry for the code puke there, but wanted you to feel a little pain as I did trying to actually implement this thing. Hopefully you didn't get tired of scrolling through that and you're still with me. ;)

It turns out that you don't have to implement the parts you don't need, but this clearly violates the Interface Segregation Principle. This interface is extremely fat and not cohesive. A better approach would have been to break it up into smaller interfaces that allow the implementers to only worry about the parts that they need. I'm not going to go into the details of splitting this up, but I think you get the idea.

Since I cannot think of another real world example, let's look at a completely bogus example. Say you have the following code:

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

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

public class Rattlesnake : Animal
{
```