Single Responsibility Principle by Jason Meridth

This post is about the first letter in Uncle Bob's SOLID acronym, Single Responsibility Principle, and a continuation of The Los Techies Pablo's Topic of the Month - March: SOLID Principles. Sean has already posted on this, but I'd like to "contribute".

***Note about the SOLID acronym and this blog "storm"**:*
This "principle" is more or less common sense, as are most of the other items in the SOLID acronym. I like the idea of this series because I personally have interviewed with companies who would ask about possible code scenarios and I respond with one of these principles or one of the GOF patterns and they look back at me with a blank stare. I know these are just labels, but if they can reduce the miscommunication possibilities and start standardizing our industry, I'm all for it. I know some of the new ideas and labels out there are still being hammered out (i.e., like the BDD discussions as of late), but that is part of the process and what has to happen in such a young industry like ours.

Single-Responsibility Principle (SRP):

A class should have only one reason to change.

A good anti-example is the Active Record pattern. This pattern is in contradiction of SRP. A domain entity handles persistence of its information. (Note: There is nothing wrong with using Active Record; I've recently used it on a quick demo site and it worked perfectly) Normally, you would have a controller method/action pass a "hydrated" entity to a method of a repository instance.

Like my favorite quote says:

Talk is cheap, show me the code ~ Linus Torvalds

Let's look at some .NET code.

Anti-SRP (Active Record)

Imagine you have a User entity that has a username and password property. I'm using the Castle Active Record libraries for this example.

```
1: using System;
2: using Castle.ActiveRecord;
3:
4: namespace ActiveRecordSample
5: {
6:     [ActiveRecord]
7:     public class User : ActiveRecordBase<User>
8:     {
9:         private int id;
10:         private string username;
11:         private string password;
```

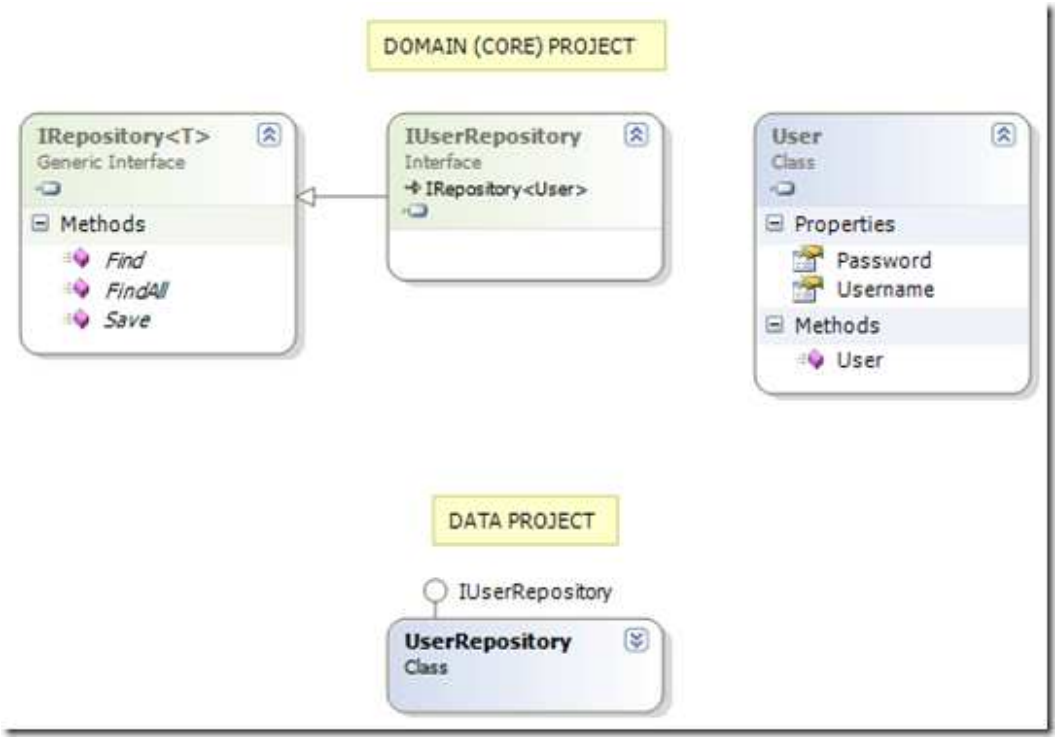
```
12:
13:     public User()
14:     {
15:     }
16:
17:     public User(string username, string password)
18:     {
19:         this.username = username;
20:         this.password = password;
21:     }
22:
23:     [PrimaryKey]
24:     public int Id
25:     {
26:         get { return id; }
27:         set { id = value; }
28:     }
29:
30:     [Property]
31:     public string Username
32:     {
33:         get { return username; }
34:         set { username = value; }
35:     }
36:
37:     [Property]
38:     public string Password
39:     {
40:         get { return password; }
41:         set { password = value; }
42:     }
43:   }
44: }
```

As you can see, you use attributes to dictate how your properties map to columns in your database table. Your entity name usually matches your table name, when using the ActiveRecord attribute with no explicit table name (i.e., [ActiveRecord("UserTableName")].

To save the user you would take an instantiated user and call user.Save(); This would cause an update to fire if the user instance had identity (aka an Id) and insert if it did not.

Translation to SRP

What I would normally do is have architecture like the following:



The UserRepository would be used by a web controller (I use monorail for my web projects), being passed a User instance, and Save(user) would be called.

```
1: using System.Collections.Generic;
2: using Castle.MonoRail.Framework;
3: using SrpPost.Core;
4: using SrpPost.Data;
5:
6: namespace SrpPost.Web.Controllers
7: {
8:     [Layout("default"), Rescue("generalerror")]
9:     public class UserController : SmartDispatcherController
10:     {
11:         private readonly IUserRepository userRepository;
12:
13:         public UserController(IUserRepository userRepository)
14:         {
15:             this.userRepository = userRepository;
16:         }
17:
18:         public void Index()
19:         {
20:             RenderView("userlist");
21:         }
22:
23:         public void Save([DataBind("user", Validate = true)] User user)
```

```
24:       {
25:           userRepository.Save(user);
26:           Flash["LoginError"] = "User saved successfully.";
27:           RenderView("userlist");
28:       }
29:   }
30: }
```

So, what it boils down to is that the user class now knows nothing on how it is persisted to the database.

SRP is one of the hardest principles to enforce because there is always room for refactoring out one class to multiple; each class has one responsibility. It is personal preference because class explosion does cause some people to become code zealots. One of my other favorite quotes lately is:

Always code as if the guy maintaining your code would be a violent psychopath and he knows where you live.

Enjoy!