**What is solid:**

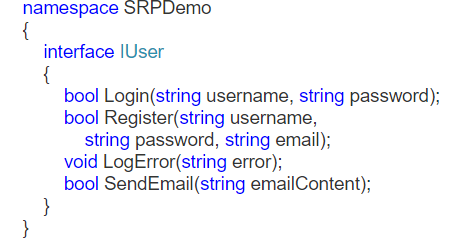
SOLID principles are the design principles that enable us to manage with most of the software design problems. These principles provide us ways to move from tightly coupled code and little encapsulation to the desired results of loosely coupled and encapsulated real needs of a business properly.

**Single Responsibility Principle:**

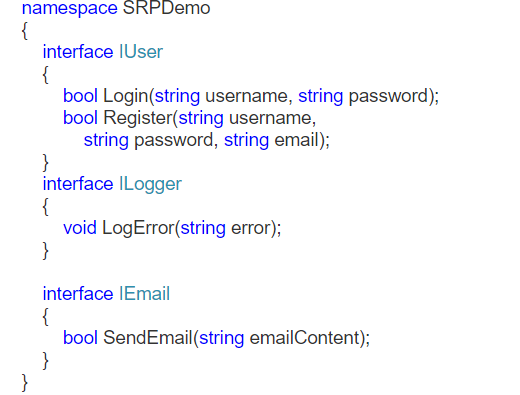
* Each class should be designed to do one thing (single responsibility.) and it should do it well.
* Each class and module should focus on a single task at a time.
* Everything in the class should be related to that single purpose.
* There can be many members in the class if they related to the single responsibility.
* With SRP, classes become smaller and cleaner.
* Code is less fragile.
* A class should have only single responsibility. A single reason to change.
* Based on separation of concerns.
* The size of the classes become shorter. This makes code easily understandable.
* SRP says a class should focus on doing one thing or have one responsibility. This doesn’t mean it should only have one method, but instead all the methods should relate to a single purpose (i.e. should be cohesive).
* A class that adheres to the SRP should be easier to change than those with multiple responsibilities. If we have calculation logic and database logic and display logic all mixed up within one class it can be difficult to change one part without breaking others.
* Mixing responsibilities also makes the class harder to understand, harder to test, and increases the risk of duplicating logic in other parts of the design

Ex:1

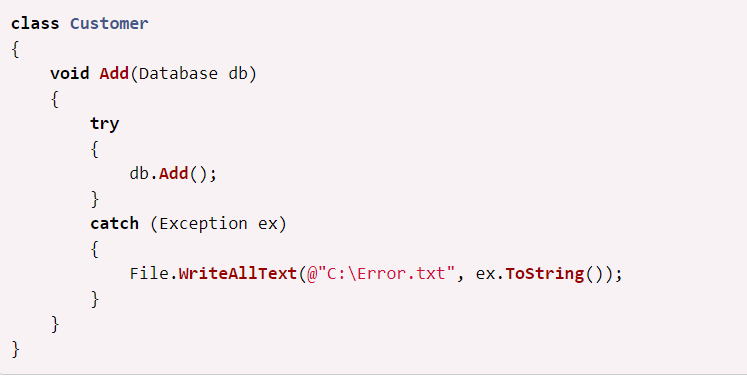
**Code before Single Responsibility Segregation:**



**Code after Single Responsibility Segregation:**



Ex:2 Customer class should do one thing only to add into db. Logging related stuff should be done by some other class.





Ex: 3

If there is a need to update XML, then a separate class should be created for the same. XmlValidator class should not be used for updating XML.

1. **public** **class** XmlValidator
2. {
3. **public** **void** Validate()
4. {
6. }
7. }

For updating a new class, it should be created.

1. **public** **class** XmlUpdate
2. {
3. **public** **void** DoUpdate()
4. {
6. }
7. }

Example: An Invoice class might have the responsibility of calculating various amounts based on its data. In that case it probably shouldn’t know about how to retrieve this data from a database, or how to format an invoice for print or display or logging, sending Email etc.

OCP:

<http://joelabrahamsson.com/a-simple-example-of-the-openclosed-principle/>

<https://www.codeproject.com/Articles/613119/SOLID-Principles-The-Open-Closed-Principle-What-Wh>

<http://csharp-video-tutorials.blogspot.com/2018/01/open-closed-principle.html?_sm_au_=iVVJKZsF5jRKCW4F>

<https://code-maze.com/open-closed-principle/>

<https://www.intertech.com/Blog/open-closed-principle-c-examples/>

<https://dotnettutorials.net/lesson/open-closed-principle/>

<https://www.dotnetcurry.com/software-gardening/1176/solid-open-closed-principle>

ISP:

<https://dzone.com/articles/solid-principles-interface-segregation-principle>

<https://www.intertech.com/Blog/the-interface-segregation-principle-with-c-examples/>

<https://www.intertech.com/Blog/the-interface-segregation-principle-with-c-examples/>

<http://csharp-video-tutorials.blogspot.com/2018/01/interface-segregation-principle.html?_sm_au_=iVVkFLL1HkbJKZrr>

**LSP:**

<https://code-maze.com/liskov-substitution-principle/>

<https://www.dotnetcurry.com/software-gardening/1235/liskov-substitution-principle-lsp-solid-patterns>

<https://www.codeproject.com/Articles/595160/Understand-Liskov-Substitution-Principle-LSP>

<https://www.c-sharpcorner.com/UploadFile/damubetha/solid-principles-in-C-Sharp/>