**Single Responsibility Principle**

The single-responsibility principle (SRP) is a computer-programming principle that states that every module, class or function in a computer program should have responsibility over a single part of that program's functionality, and it should encapsulate that part. All of that module, class or function's services should be narrowly aligned with that responsibility.

(Wikipedia)

A class should have only one reason to change.

- Robert C Martin, Agile Software Develoment

Martin described it as being based on the principle of cohesion.

Martin defines a responsibility as a reason to change, and concludes that a class or module should have one, and only one, reason to be changed (i.e. rewritten).

**Guideline:**

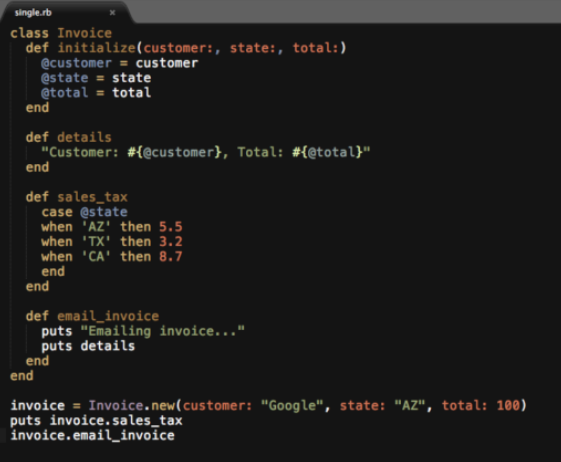
*Prefer cohesive software entities. Everything that does not strictly belong together, should be separated.*

**Key Points:**

* Each module and class should focus on a single task at a time
* Everything in the class should be related to that single purpose
* There could be many members in the class as long as they related to single responsibility
* With SRP, classes become smaller and cleaner
* Code is less fragile

# Rule of Thumb: No ‘Ands’ Allowed

When it comes to following the SOLID design pattern a good rule of thumb is that if your description of a class has the word **and** in it, then it may need to be refactored. For example, let’s describe this class:



THE INVOICE CLASS PRINTS OUT INVOICE DETAILS AND CALCULATES SALES TAX AND EMAILS THE INVOICE.

Whenever I’m performing a refactor I like to treat the behavior between the ANDs as their own class.

# Example

Consider a module that compiles and prints a report. Imagine such a module can be changed for two reasons.

1. The content of the report could change.
2. The format of the report could change.

These two things change for very different causes; one substantive, and one cosmetic.

The single responsibility principle says that these two aspects of the problem are really two separate responsibilities, and should therefore be in separate classes or modules. It would be a bad design to couple two things that change for different reasons at different times.

The reason it is important to keep a class focused on a single concern is that it makes the class more robust.

The Single Responsibility Principle (SRP) states that each software module should have one and only one reason to change.

**What defines a reason to change?**

Certainly the code is not responsible for bug fixes or refactoring. Those things are the responsibility of the programmer, not of the program. But if that is the case,

**Who must the design of the program respond to?**

Imagine a typical business organization. There is a CEO at the top. Reporting to that CEO are the C-level executives: the CFO, COO, and CTO among others.

Now consider this bit of Java code:

public class Employee {

public Money calculatePay();

public void save();

public String reportHours();

}

CFO is responsible for calculatePay

CTO is responsible for save

COO is responsible for reportHours

So it stands to reason that when changes are made to the algorithm within the calculatePay method, the request for those changes will originate from the organization headed by the CFO. Similarly it will be the COO’s organization that will request changes to the reportHours method, and the CTOs organization that will request changes to the save method.

And this gets to the crux of the Single Responsibility Principle. This principle is about people.

This is the reason we do not put SQL in JSPs. This is the reason we do not generate HTML in the modules that compute results. This is the reason that business rules should not know the database schema. This is the reason we separate concerns.

Another wording for the Single Responsibility Principle is:

Gather together the things that change for the same reasons. Separate those things that change for different reasons.

This is just another way to define cohesion and coupling. We want to increase the cohesion between things that change for the same reasons, and we want to decrease the coupling between those things that change for different reasons.

However, as you think about this principle, remember that the reasons for change are people.

# Example

Below example is an interface for user registration, login, error logging and email sent.

class IUser { // Mixed Interface for register, login, error logging and sent email

bool Register(string username, string password, string emailid) = 0;

bool Login(string username, string password) = 0;

void LogError(string error) = 0;

void send\_email(string email\_content) = 0;

};

Above interface has more than one responsibility and violates SRP principle.

We can separate interface of user register and login from error logging and email sent.

class IUser { // Interface for User register and login

bool Register(string username, string password, string emailid) = 0;

bool Login(string username, string password) = 0;

};

class ILogger { // Interface for logging

void LogError(string error) = 0;

};

class Iemail { // Interface for sent email

void send\_email(string email\_content) = 0;

};

# References

<https://en.wikipedia.org/wiki/Single-responsibility_principle>

<https://blog.cleancoder.com/uncle-bob/2014/05/08/SingleReponsibilityPrinciple.html>

<https://www.crondose.com/2016/08/solid-development-single-responsibility-principle/>

<https://www.youtube.com/watch?v=Ntraj80qN2k>

<https://www.youtube.com/watch?v=hGf2upfDpdo&list=PL6n9fhu94yhXjG1w2blMXUzyDrZ_eyOme&index=2>

<https://web.archive.org/web/20150202200348/http://www.objectmentor.com/resources/articles/srp.pdf>



# END