Single Responsibility Principle

Object-Oriented Terminology

In object-oriented programming (Java, among other languages, follows this paradigm), you will often hear terms such as robustness, cohesion, coupling etc. **Cohesion** is a way to measure how much the code segments within one module (methods of a class, classes inside a package…) belong together. The higher the cohesion – the better, since high cohesion implies easier maintenance and debugging, greater code functionality and reusability. The term cohesion is sometimes contrasted with the concept of **coupling**, and often, loose coupling of modules is related to high cohesion.

Another widely used term is **robustness,** which could be defined as the ability of a computer system or algorithm to handle mistakes and malfunctions (which could be caused by various factors such as programmer's mistake or incorrectly formatted user input). A robust system is one that can handle these unwanted situations elegantly. There are various ways for a software engineer to achieve robustness, such as testing the code for different kinds of inputs, but generally, in order to achieve robustness (and high cohesion), programmers follow a certain set of rules and principles for better organization of object-oriented programs. One such principle is the single responsibility principle.

Single Responsibility Principle

The single responsibility principle revolves around the claim that a certain code module (most often, a class) should only have responsibility over one part of the functionality provided by the software. In software engineering books, this is sometimes also defined like this: the module should only have **one reason to change**. This means that a division of concerns is performed in the program, and the methods for every concern should be completely encapsulated by a single class. Now it is obvious that this approach contributes to the high cohesion – since methods related to the same concern (same part of the functionality) will be members of the same class, and robustness – since this reduces the possibility of error. Furthermore, if an error does occur, the programmer will be more likely to find the cause, and finally, solve the problem.

The single responsibility principle is founded on one of the basic, general ideas of object-oriented programming – the so-called *divide and conquer* principle – solving a problem by solving its multiple sub-problems. This approach prevents the creation of *"God objects"* – objects that "*know too much or do too much*".

The classes you write, should not be a swiss army knife. They should do one thing, and to that one thing well.

SINGLE RESPONSIBILITY PRINCIPLE
Every object should have a single responsibility, and all its services should be narrowly aligned with that responsibility.

Friends don't let friends code like this.

(Bad) Example

Let's consider this classic example in Java – "objects that can print themselves".

```java
class Text {
    String text;
    String author;
    int length;

    String getText() { ... }
    void setText(String s) { ... }
    String getAuthor() { ... }
    void setAuthor(String s) { ... }
    int getLength() { ... }
    void setLength(int k) { ... }

    /*methods that change the text*/
    void allLettersToUpperCase() { ... }
    void findSubTextAndDelete(String s) { ... }

    /*method for formatting output*/
    void printText() { ... }
}
```

At first glance, this class might look correctly written. However, it contradicts the single responsibility principle, in that it has multiple reasons to change: we have two methods which change the text itself, and one which prints the text for the user. If any of these methods is called, the class will change. This is also not good because it mixes the logic of the class with the presentation.

Better Example

One way of fixing this is writing another class whose only concern is to print text. This way, we will separate the functional and the “cosmetic” parts of the class.

```
class Text {
    String text;
    String author;
    int length;

    String getText() { ... }
    void setText(String s) { ... }
    String getAuthor() { ... }
    void setAuthor(String s) { ... }
    int getLength() { ... }
    void setLength(int k) { ... }

    /*methods that change the text*/
    void allLettersToUpperCase() { ... }
    void findSubTextAndDelete(String s) { ... }
}

class Printer {
    Text text;

    Printer(Text t) {
        this.text = t;
    }

    void printText() { ... }
}
```

Summary

In the second example we have divided the responsibilities of editing text and printing text between two classes. You can notice that, if an error occurred, the debugging would be easier, since it wouldn't be that difficult to recognize where the mistake is. Also, there is less risk of accidentally introducing software bugs, since you're modifying a smaller portion of code.

Even though it's not that noticeable in this example (since it is small), this kind of approach allows you to see the "bigger picture" and not lose yourself in the code; it makes programs easier to upgrade and expand, without the classes being too extensive, and the code becoming confusing.

Single Responsibility Principle in Spring

As you become more comfortable using Spring components and coding to support Inversion of Control and Dependency Injection in Spring, you will find your classes will naturally adhere to the single responsibility principle. A typical violation of the single responsibility principle I often see in legacy Spring applications is an abundance of code in controller actions. I've seen Spring controllers getting JDBC connections to make calls to the database. This is a clear violation of the single responsibility principle. Controller objects have no business interacting with the database. Nor do controllers have any business implementing other business logic. In practice your controller methods should be very simple and light. Database calls and other business logic belong in a service layer.

13 comments on "Single Responsibility Principle"

KAPIL April 11, 2016 at 9:10 am # Reply

good explanation John 🙂

Richard Langlois May 11, 2016 at 12:30 pm # Reply

I really like when you put emphasis about what should be the responsibilities of the Controller, service, and repository. This is also very much in line with what Spring annotations offer: @Controller, @Service, @Repository.

Erikson Rodriguez November 18, 2016 at 12:02 pm # Reply

Good tip, and very usefull in 100% of the case!

---

vivekanandan November 22, 2016 at 9:15 am # Reply

Good explanation!

Harinath November 23, 2016 at 9:27 pm # Reply

Clear and concise about SOLID

---

carlos lopez November 28, 2016 at 10:41 pm # Reply

Nice!!

José Pinto February 22, 2017 at 6:04 pm # Reply

As a recent adopter of spring, I find this to be something important and even most to keep in mind at all times.
Good post!

Dustin W March 1, 2017 at 9:34 am # Reply

As a new developer, I am a little confused at why you would abstract the printText method and not the other two methods. Could you have created an AlterText class that would handle all the ways text can be changed and left the printText method inside the Text class (like toString?)… Would this still satisfy the single responsibility principle?

I would like to also say that other than this one question, you do a great job of simplifying things and as a young… err… newer developer, it really helps. And your breadth of knowledge and willingness to share it is beyond helpful. I feel like I have spent a week combing through your informative, but succinct articles.

jt March 1, 2017 at 10:42 am # Reply

Thanks Dustin! Glad you like my content.

The example was only meant to show the concept of refactoring to support SRP.

Bruno D March 30, 2017 at 10:08 am # Reply

Awesome. Thanks for the article !

dougiet91 November 11, 2017 at 9:25 am # Reply

This is really good. Love the explanation.

Jasvir December 28, 2017 at 5:21 am # Reply

Nice example.

Segregating the responsibilities to individual classes each of which would address a set of related functions should be the primary thought while designing and system. SRP together with other design principles go a long way in creating long standing software designs.

Jochen March 27, 2018 at 8:12 am # Reply

Surely you can delete my comments; I only like to tell you typing-errors I found, perhaps you like to correct it: “we have tho methods” – go for “two” 😉