Single Responsibility Principle

> A class should have just one reason to change.

Try to make every class responsible for a single part of the functionality provided by the software, and make that responsibility entirely encapsulated by (you can also say *hidden within*) the class.

The main goal of this principle is reducing complexity. You don't need to invent a sophisticated design for a program that only has about 200 lines of code. Make a dozen methods pretty, and you'll be fine.

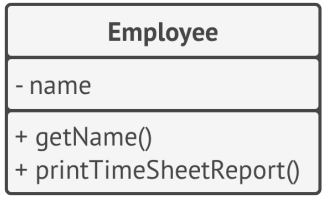
The real problems emerge when your program constantly grows and changes. At some point classes become so big that you can no longer remember their details. Code navigation slows down to a crawl, and you have to scan through whole classes or even an entire program to find specific things. The number of entities in program overflows your brain stack, and you feel that you're losing control over the code.

There's more: if a class does too many things, you have to change it every time one of these things changes. While doing that, you're risking breaking other parts of the class which you didn't even intend to change.

If you feel that it's becoming hard to focus on specific aspects of the program one at a time, remember the single responsibility principle and check whether it's time to divide some classes into parts.

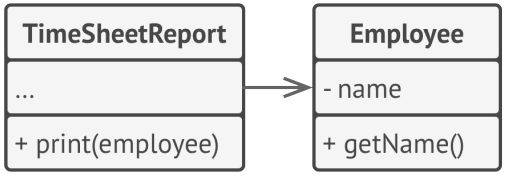
Example

The `Employee` class has several reasons to change. The first reason might be related to the main job of the class: managing employee data. However, there's another reason: the format of the timesheet report may change over time, requiring you to change the code within the class.



BEFORE: the class contains several different behaviors.

Solve the problem by moving the behavior related to printing timesheet reports into a separate class. This change lets you move other report-related stuff to the new class.



AFTER: the extra behavior is in its own class.