**Example of the Single Responsibility Principle**

Let’s see an example. I’ll use Java but you can apply SOLID design principles to any other OOP languages, too.

Say, we are writing a Java application for a book store. We create a Book class that lets users get and set the titles and authors of each book, and search the book in the inventory.

class Book {

String title;

String author;

String getTitle() {

return title;

}

void setTitle(String title) {

this.title = title;

}

String getAuthor() {

return author;

}

void setAuthor(String author) {

this.author = author;

}

void searchBook() {...}

}

However, the above code violates the Single Responsibility Principle, as the Book class has two responsibilities. First, it sets the data related to the books (title and author). Second, it searches for the book in the inventory. The setter methods change the Book object, which might cause problems when we want to search the same book in the inventory.

To apply the Single Responsibility Principle, we need to decouple the two responsibilities. In the refactored code, the Book class will only be responsible for getting and setting the data of the Book object.

class Book {

String title;

String author;

String getTitle() {

return title;

}

void setTitle(String title) {

this.title = title;

}

String getAuthor() {

return author;

}

void setAuthor(String author) {

this.author = author;

}

}

Then, we create another class called InventoryView that will be responsible for checking the inventory. We move the searchBook() method here and reference the Book class in the constructor.

class InventoryView {

Book book;

InventoryView(Book book) {

this.book = book;

}

void searchBook() {...}

}

On the UML diagram below, you can see how the architecture changed after we refactored the code following the Single Responsibility Principle. We split the initial Book class that had two responsibilities into two classes, each having its own single responsibility.

