# Exercise: Encapsulation and Polymorphism

This document defines an in-class exercise from the ["OOP" Course @ Software University](https://softuni.bg/courses/oop/).

## Cohesion and Coupling

Maintaining strong cohesion and keeping coupling between classes loose are two of the most important principles of high-quality code. You’re given a program which needs refactoring to follow these principles. You’ll be working with the **CohesionAndCoupling.sln** solution. The application contains an engine which executes commands and prints their result on the console. Follow the steps below to improve code quality.

### Step 1. Get to Know the Application

Before applying any changes to the code, you need to **study** it and figure out how it works. The application models a book store with methods for **adding**, **selling** and **removing** books. Each command is executed and a result is returned as a string and printed on the console. Check out the provided classes.

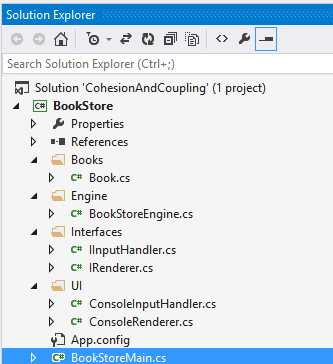
### Step 2. Single Responsibility

**Strong cohesion** means that a class/method is responsible for one specific task. There is one class and a method in the engine which break this principle.

1. The **UserInterface** class does two different things – reads user input and prints output. What happens if we want output to be printed to a file? We’ll need a new class that reads input from the console and writes to a file. For any combination of user input source and output there has to be a new class, which is cumbersome.

Following the principle of strong cohesion, extract **two interfaces** – **IRenderer** with method **WriteLine**, and **IInputHandler** with method **ReadLine**. Add two classes – **ConsoleRenderer** implementing IRenderer and **ConsoleInputHandler** implementing IInputHandler. Now, if we want the output to go to a file, we can just add a new class FileRenderer implementing the IRenderer interface.

Structure:

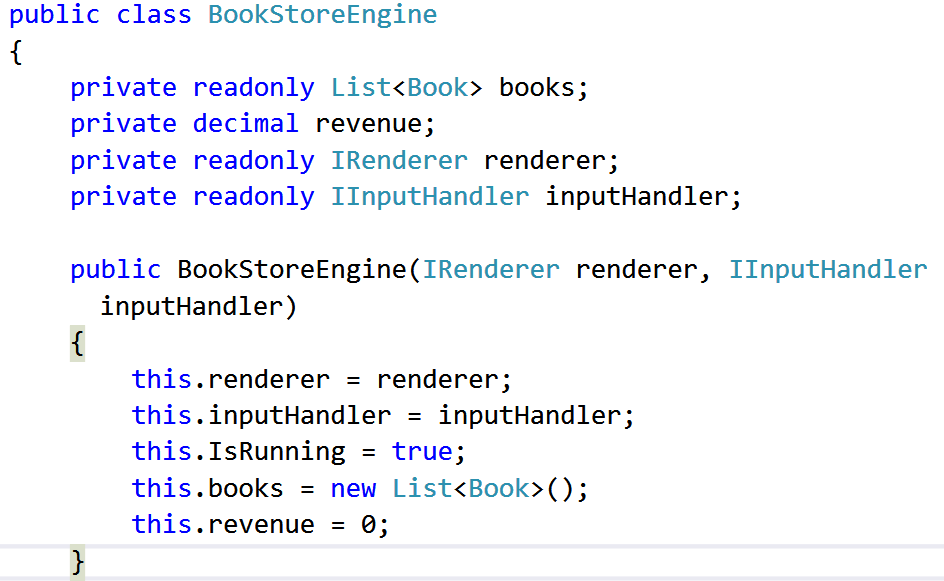


1. In the engine, there is a method called **ExecuteRemoveSellBookCommand**, which does two different things. Separate it into two methods – **ExecuteRemoveBookCommand** and **ExecuteSellBookCommand**.

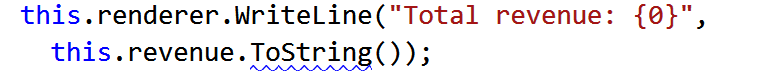
### Step 3. Loose Coupling

Currently, the engine is **coupled** with two concrete classes – **Console** and **Book**.

1. Decoupling the class from the console can be done through **dependency injection**. The engine needs a way to take user input and print stuff, in other words, it needs an **IRenderer** and an **IInputHandler** to perform these tasks. Create **two private fields** in the engine, one will hold an IRenderer and the other an IInputHander. In the Engine’s constructor, add two parameters – renderer and inputHandler; when instantiating an engine the user will have to provide a renderer and input handler.

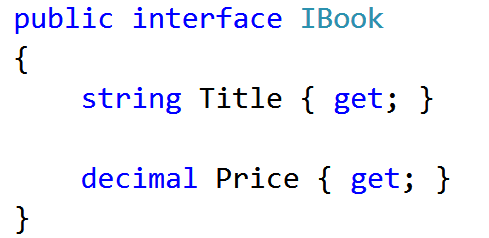


Create a console renderer and console input handler in the main program and pass them to the engine’s constructor. Anywhere in the code where you see **Console.ReadLine** replace it with **this.inputHandler.ReadLine**, and anywhere you see **Console.WriteLine** exchange it with **this.renderer.WriteLine**. Now, if we want to print to a file, we have to create a class FileRenderer, instantiate it in the main program (with a file path) and pass it to the engine’s constructor; no other modifications will be necessary.

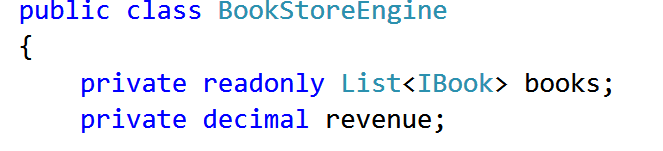


1. What happens if we want to add different types of books? To make the application more flexible, we can extract an interface **IBook**. Let’s say that whatever is being sold at the bookstore needs to have at least a **title** and **price**. Create the interface; the Book class should implement it. Now any method or list accepting Book objects should be modified to accept **IBook** instead.

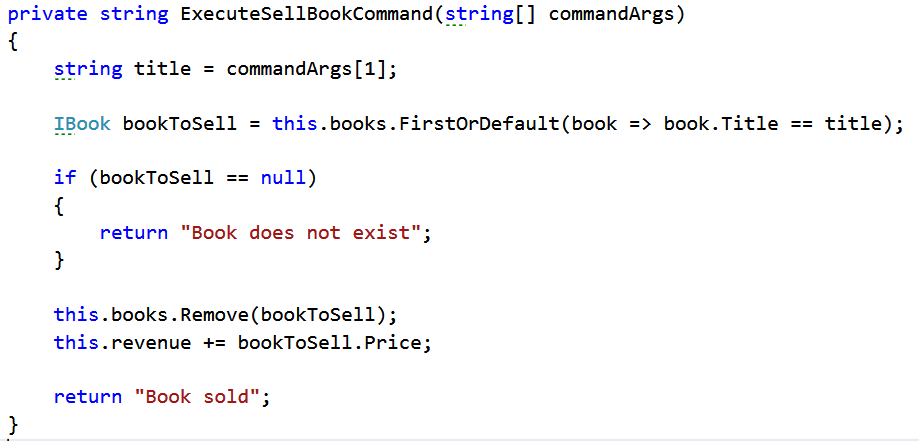
Interface:



Engine fields:

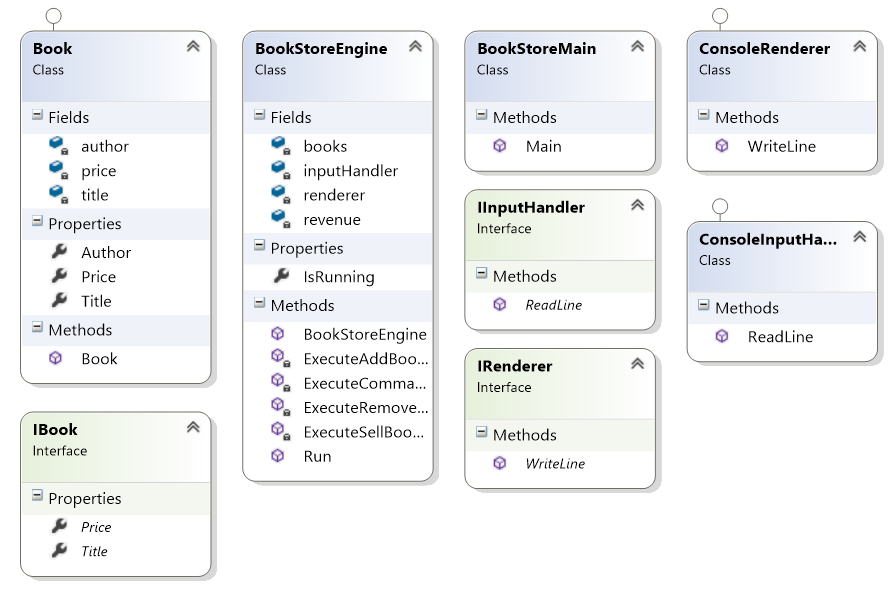


ExecuteSellBookCommand method:



### Step 4. View Hierarchy and Test

This is how the class diagram should look like:



You can use the following commands to test the program:

|  |  |
| --- | --- |
| **Input** | **Output** |
| add Game\_Of\_Thrones GRR\_Martin 12.90  add Pod\_Igoto Ivan\_Vazov 4.45  remove Clash\_Of\_Kings  sell Pod\_Igoto  remove Pod\_Igoto  remove Game\_Of\_Thrones  sell Game\_Of\_Thrones  stop | Book added  Book added  Book does not exist  Book sold  Book does not exist  Book removed  Book does not exist  Goodbye!  Total revenue: 4.45 |