Graphical User Interfaces

Design Fundamentals

COMP2603
Object Oriented Programming 1

Week 7, Lecture 1

Outline

- Cohesion
- Coupling
- Three Tier Architecture
- Model View Separation

Cohesion

Cohesion concerns the inner strength of a class, i.e., how strongly related the parts of a class are.

A class whose parts are strongly related to each other and to the concept being represented is said to be strongly cohesive.

Strongly cohesive classes are easier to understand, debug and maintain.

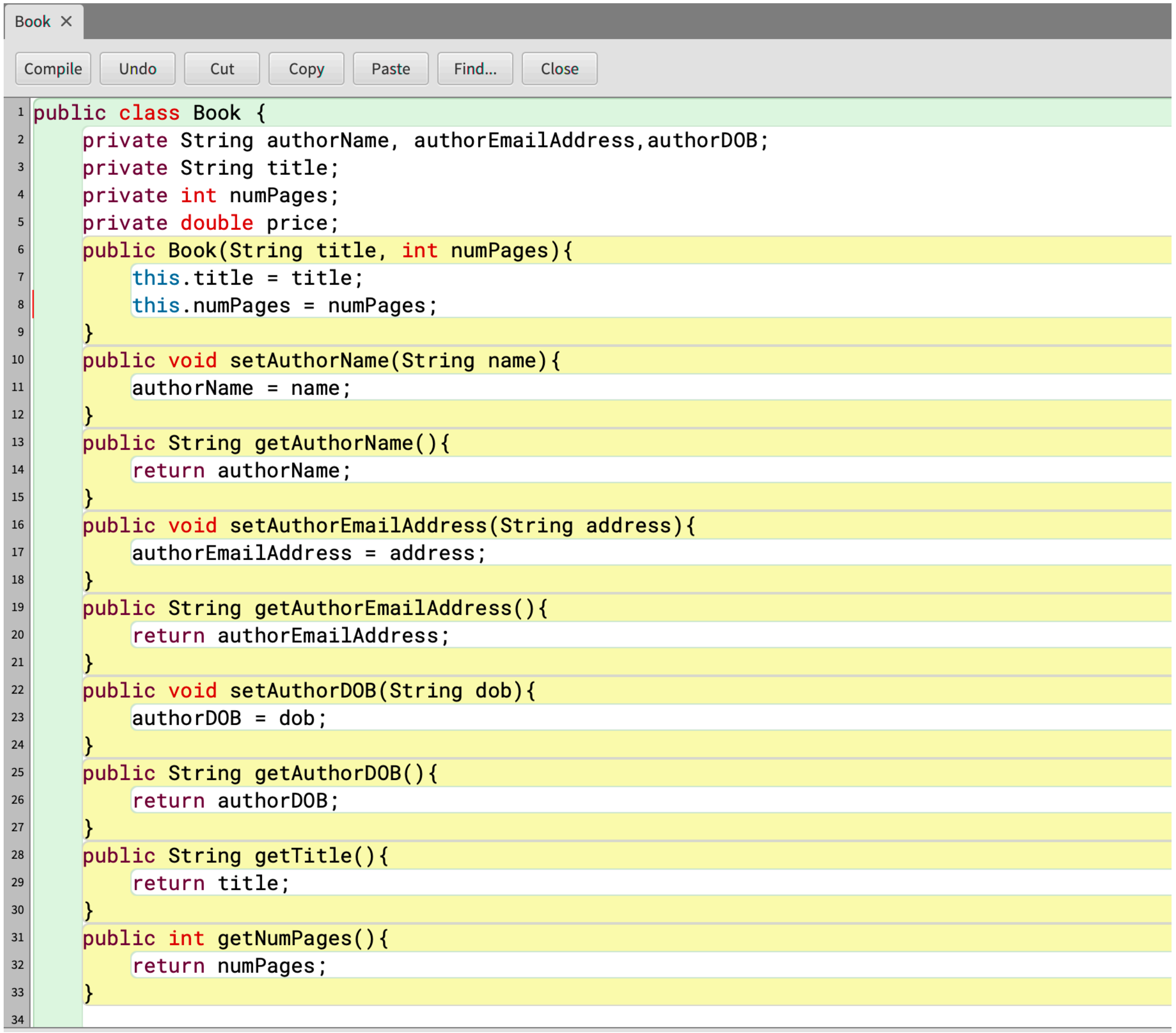
Weak Cohesion

In contrast, a weakly cohesive class is one whose parts are hardly related to each other.

These classes tend to embody more than one unrelated concept and takes on responsibilities that could be best handled by another class.

Eg. If the Book class stores detailed author information such as telephone number, author name, address etc.

Example - Weak Cohesion

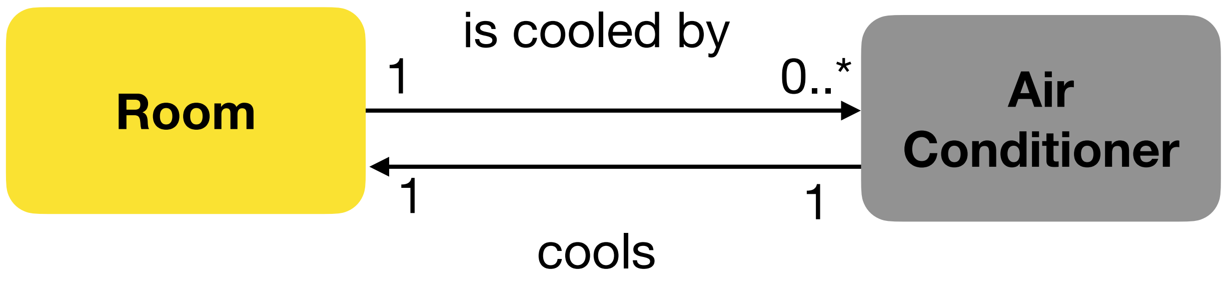


```java
public class Book {
    private String authorName, authorEmailAddress,authorDOB;
    private String title;
    private int numPages;
    private double price;
    public Book(String title, int numPages){
        this.title = title;
        this.numPages = numPages;
    }
    public void setAuthorName(String name){
        authorName = name;
    }
    public String getAuthorName(){
        return authorName;
    }
    public void setAuthorEmailAddress(String address){
        authorEmailAddress = address;
    }
    public String getAuthorEmailAddress(){
        return authorEmailAddress;
    }
    public void setAuthorDOB(String dob){
        authorDOB = dob;
    }
    public String getAuthorDOB(){
        return authorDOB;
    }
    public String getTitle(){
        return title;
    }
    public int getNumPages(){
        return numPages;
    }
```

Object Interactions

An object-oriented program consists of a set of objects that collaborate to achieve some goal.

Two objects collaborate when one object requests a service from the other.



Coupling

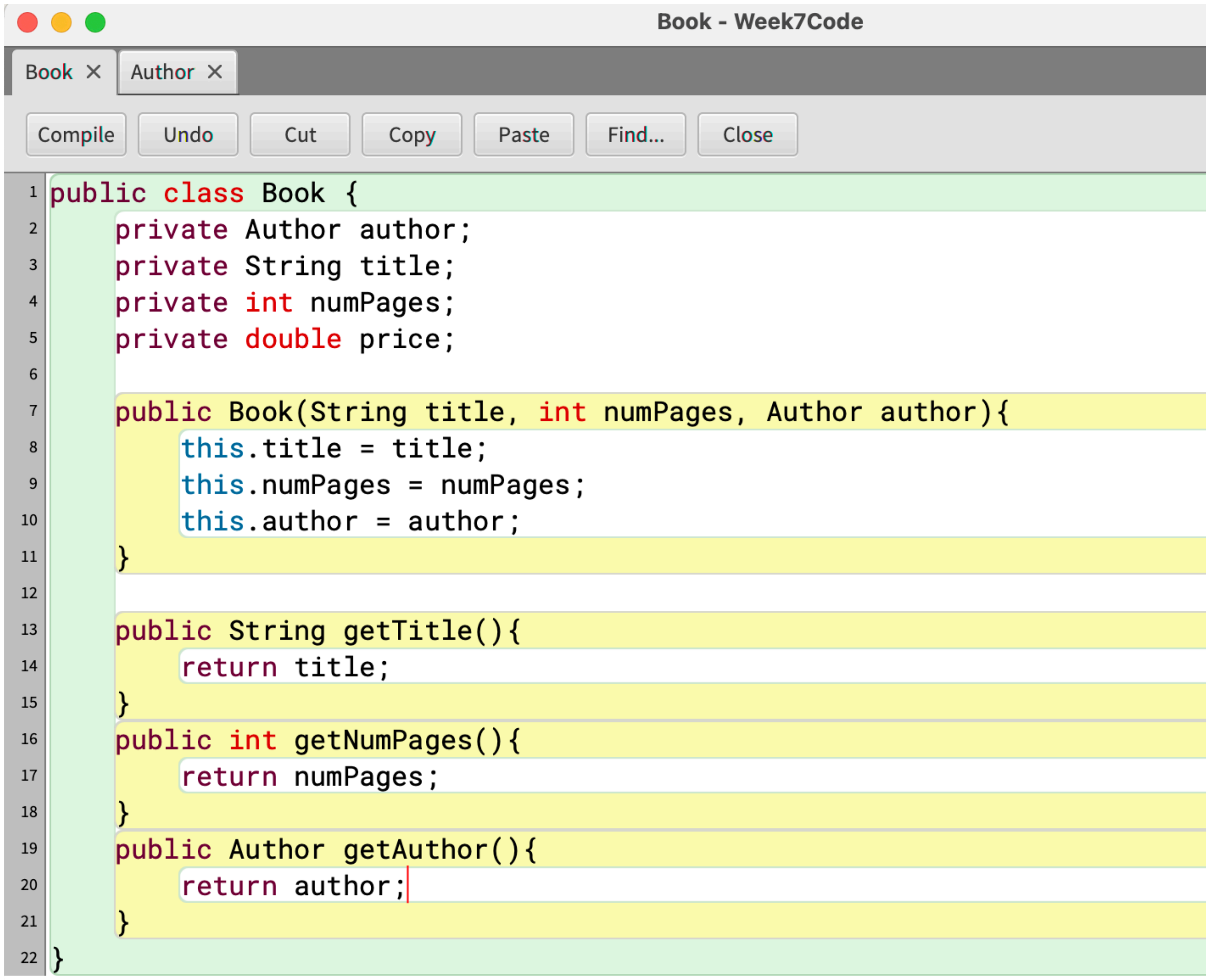
Object-Oriented coupling refers to the degree or strength of interconnection among classes.

Loosely coupled classes are desirable since each class can be handled in a relatively independent manner.

Tightly coupled classes are not desirable.

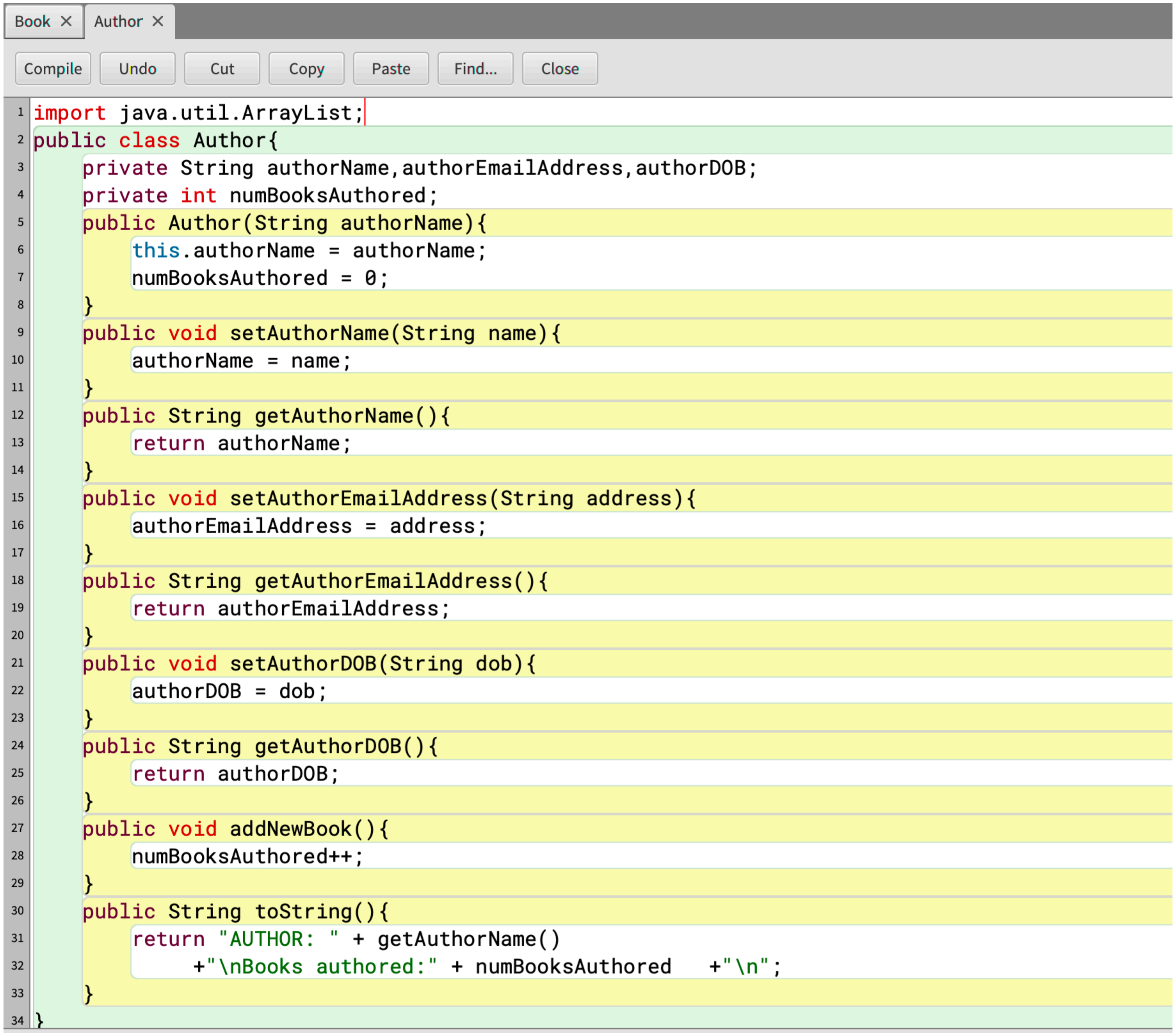
A class can become coupled to another class if it knows about that class in some way.

Example - Book.java



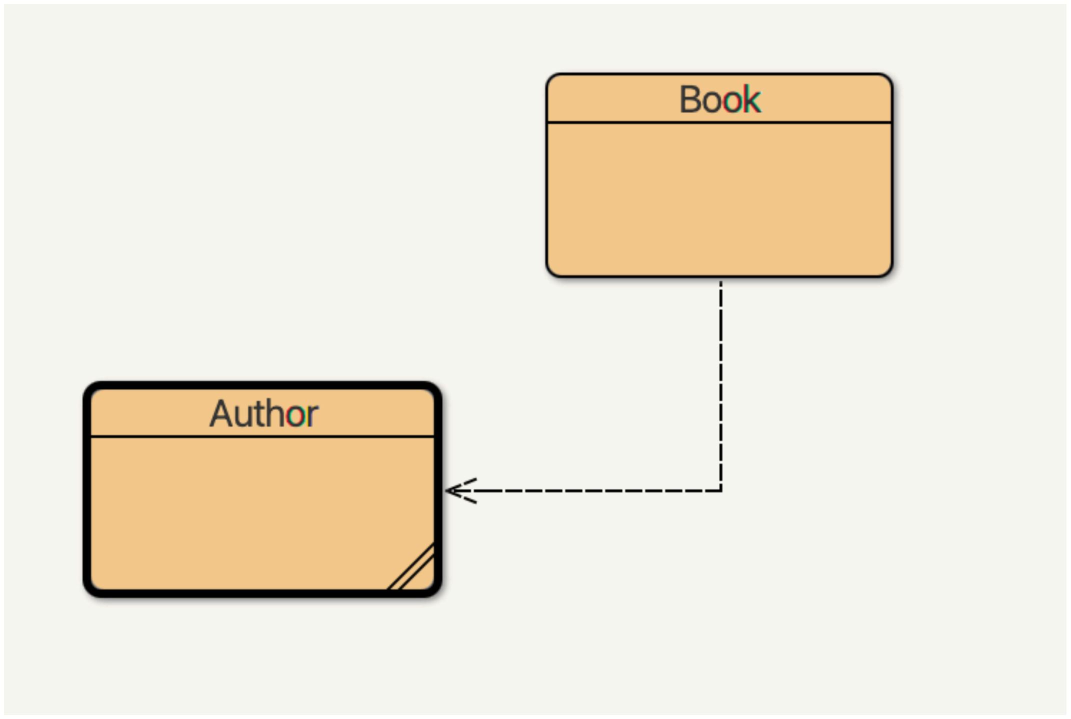
```java
public class Book {
	private Author author;
	private String title;
	private int numPages;
	private double price;

	public Book(String title, int numPages, Author author){
		this.title = title;
		this.numPages = numPages;
		this.author = author;
	}

	public String getTitle(){
		return title;
	}
	public int getNumPages(){
		return numPages;
	}
	public Author getAuthor(){
		return author;
	}
}
```

Example - Author.java

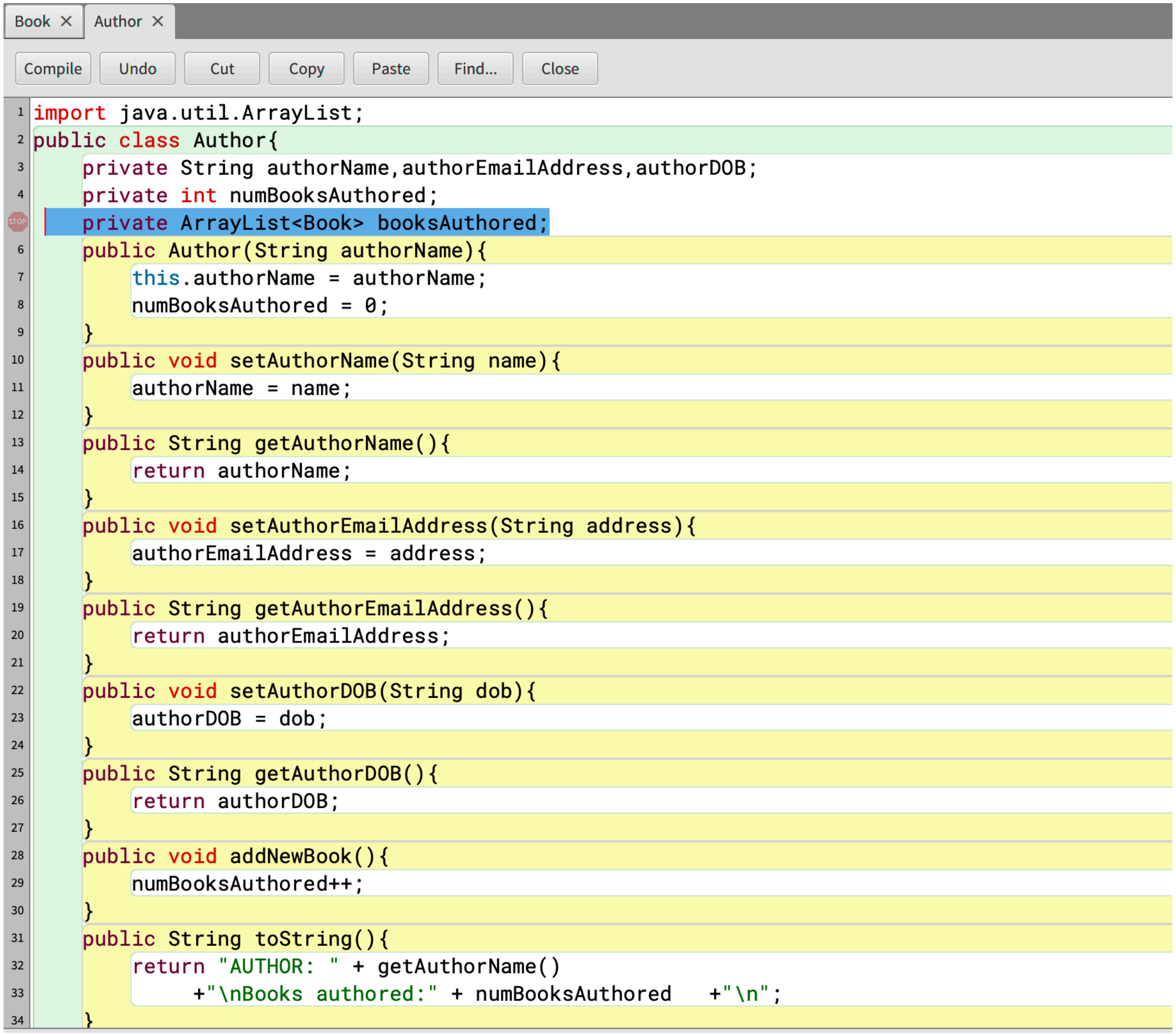


```java
import java.util.ArrayList;
public class Author{
    private String authorName,authorEmailAddress,authorDOB;
    private int numBooksAuthored;
    public Author(String authorName){
        this.authorName = authorName;
        numBooksAuthored = 0;
    }
    public void setAuthorName(String name){
        authorName = name;
    }
    public String getAuthorName(){
        return authorName;
    }
    public void setAuthorEmailAddress(String address){
        authorEmailAddress = address;
    }
    public String getAuthorEmailAddress(){
        return authorEmailAddress;
    }
    public void setAuthorDOB(String dob){
        authorDOB = dob;
    }
    public String getAuthorDOB(){
        return authorDOB;
    }
    public void addNewBook(){
        numBooksAuthored++;
    }
    public String toString(){
        return "AUTHOR: " + getAuthorName()
            +"\nBooks authored:" + numBooksAuthored   +"\n";
    }
}
```

Example - Loose Coupling

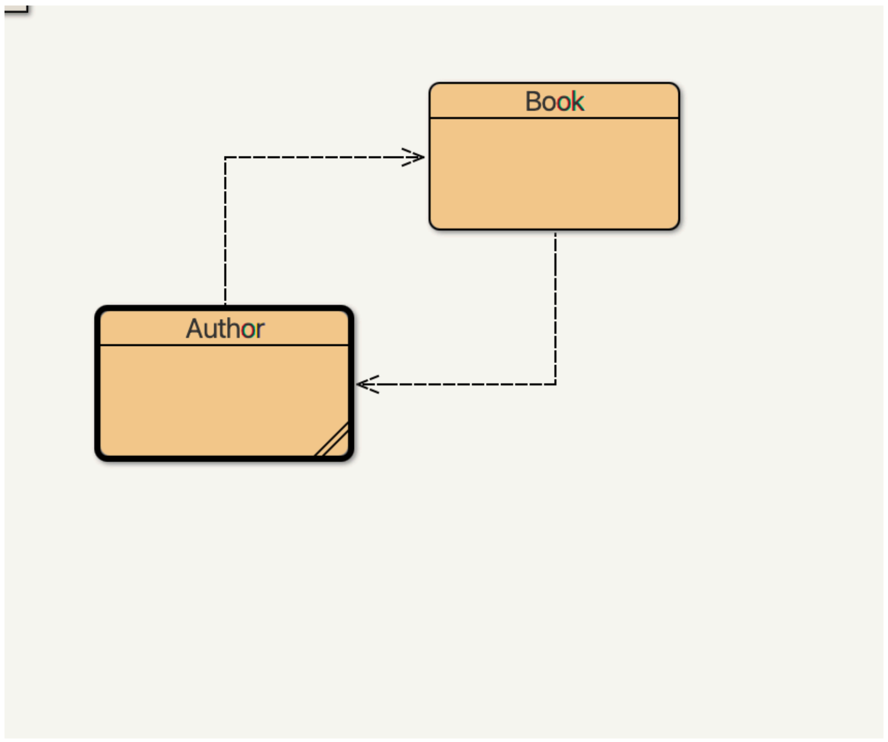


Example - Author.java



```java
import java.util.ArrayList;
public class Author{
    private String authorName,authorEmailAddress,authorDOB;
    private int numBooksAuthored;
    private ArrayList<Book> booksAuthored;
    public Author(String authorName){
        this.authorName = authorName;
        numBooksAuthored = 0;
    }
    public void setAuthorName(String name){
        authorName = name;
    }
    public String getAuthorName(){
        return authorName;
    }
    public void setAuthorEmailAddress(String address){
        authorEmailAddress = address;
    }
    public String getAuthorEmailAddress(){
        return authorEmailAddress;
    }
    public void setAuthorDOB(String dob){
        authorDOB = dob;
    }
    public String getAuthorDOB(){
        return authorDOB;
    }
    public void addNewBook(){
        numBooksAuthored++;
    }
    public String toString(){
        return "AUTHOR: " + getAuthorName()
            +"\nBooks authored:" + numBooksAuthored   +"\n";
    }
```

Example - Tighter Coupling



Loose Coupling

Loosely coupled classes facilitate:

- The replacement of one class by another so that only a few classes are affected by the change.
- The speedy debugging of errors since it is easier to track down an error and isolated the defective class causing the error.

Three-Tier Architecture for Object Oriented Software

Typically, object-oriented programs consist of several layers of software.

Each layer serves a particular purpose.

It is common to use a three-layered architecture when designing these kinds of programs.

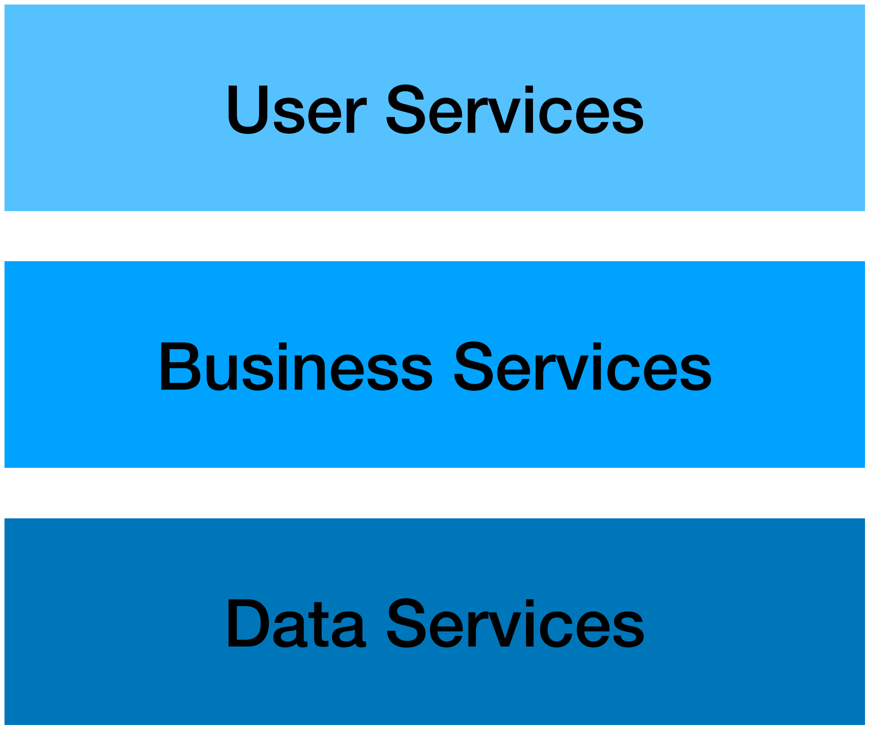
This is known as the **Three-Tiered Architecture**

Three-Tiered Architecture

The Three-Tiered Architecture consists of 3 vertical layers:

1. **User Services:** visual interface for presenting information and gathering data
2. **Business Services:** tasks and rules that govern application processing
3. **Data Services:** persistent storage mechanism to maintain, access and update data

Layers in the Three-Tier Architecture



Model View Separation

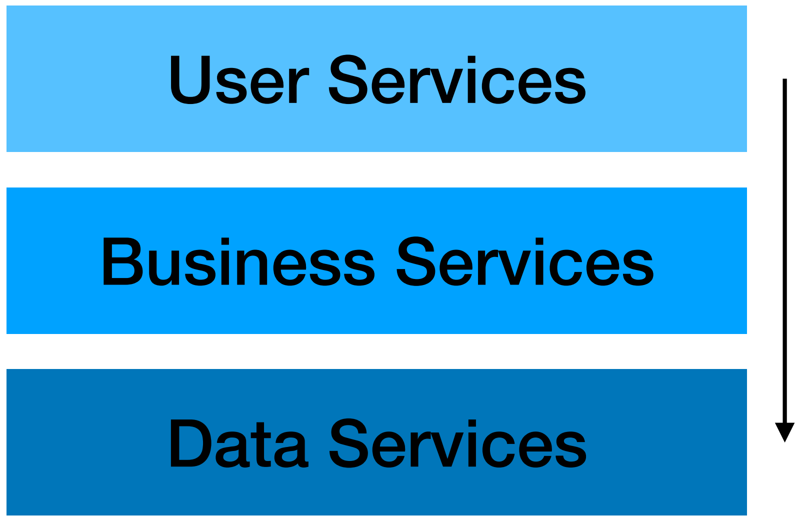
The Model-View Separation design pattern states that model object (domain objects or objects in the Business Services tier) should not have direct knowledge of or be directly coupled to view objects (user interface objects).

Communication between Domain Objects and User Interface Objects

In order for the objects in the user services tier to display information, methods must be invoked on the relevant domain objects.

The data is then displayed using GUI components.

This approach is called **pull-from-above.**

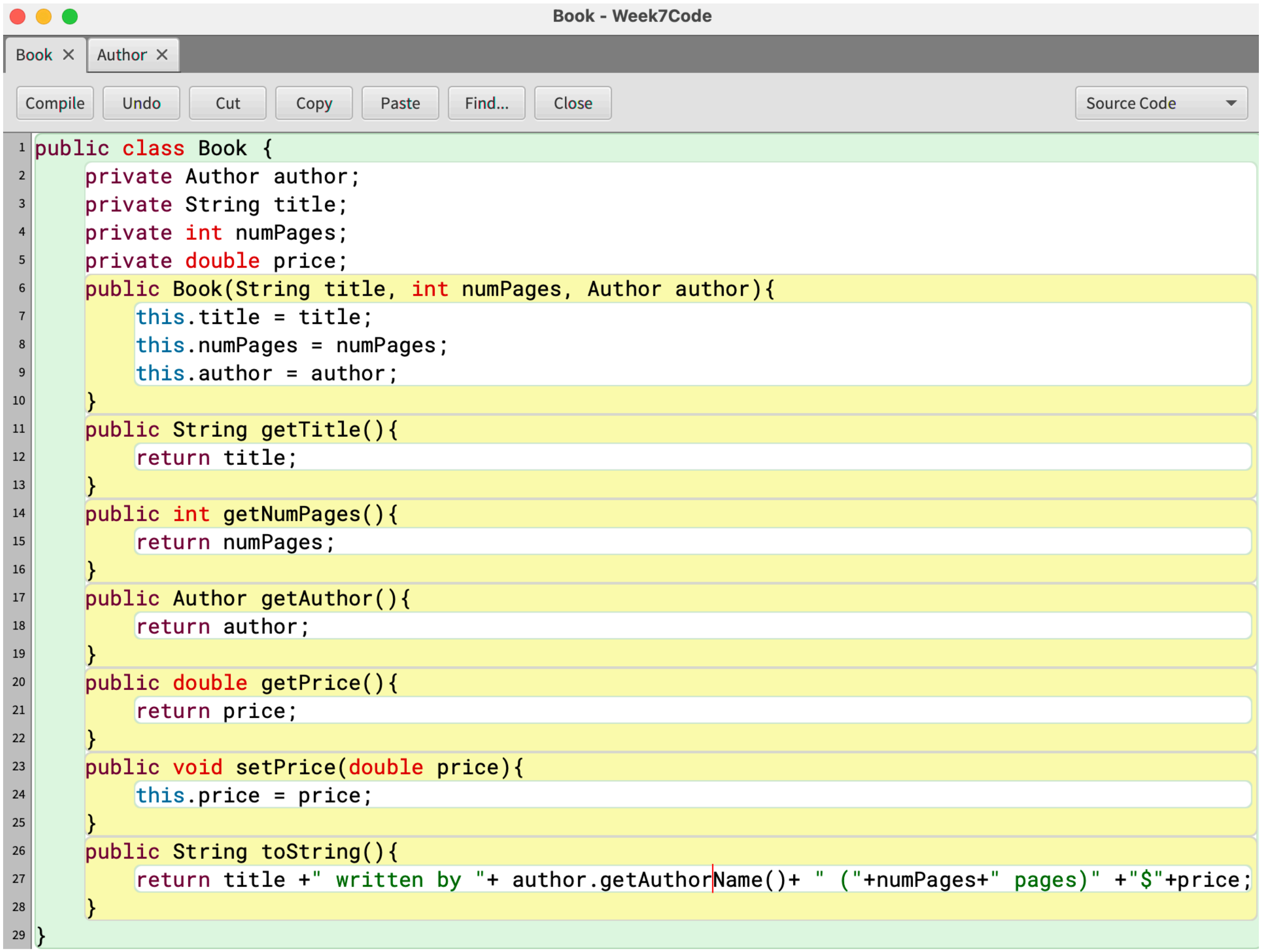


Advantages of Model-View Separation

Advantages of using the Model-View Separation:

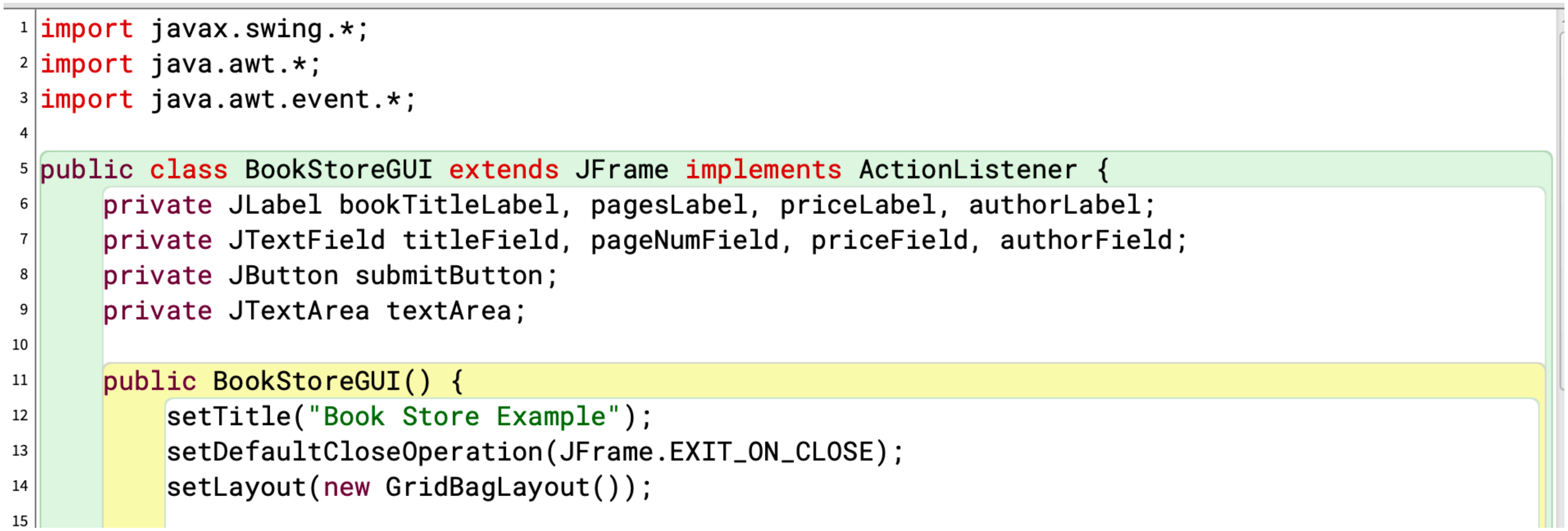
- Model can be developed independently of the user interface
- New views can be easily connected to an existing model without affecting the objects in the model
- Multiple, simultaneous views of the same model can be developed (desktop, smart phone, web form views)
- The classes in the model can be easily ported to another user interface

Example - Book.java



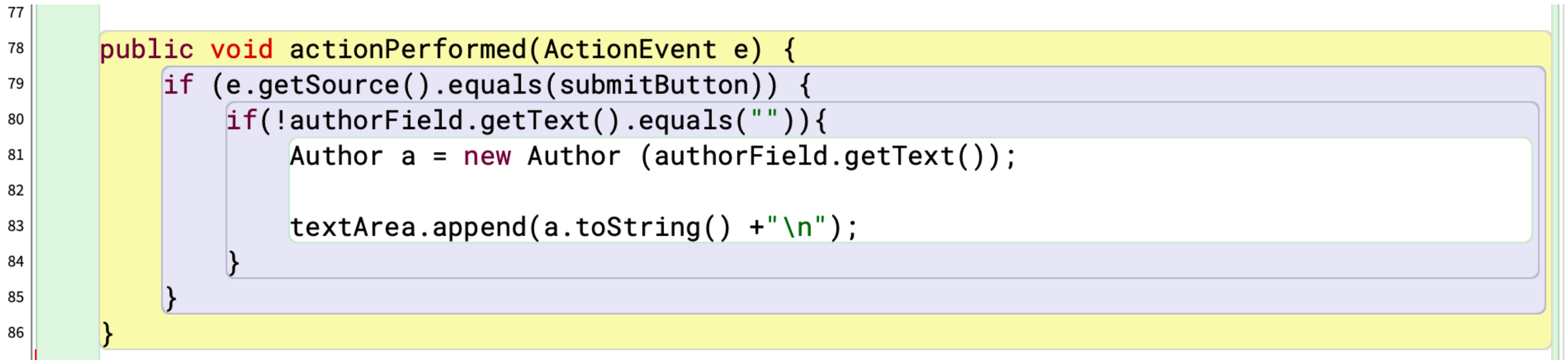
```java
public class Book {
	private Author author;
	private String title;
	private int numPages;
	private double price;
	public Book(String title, int numPages, Author author){
		this.title = title;
		this.numPages = numPages;
		this.author = author;
	}
	public String getTitle(){
		return title;
	}
	public int getNumPages(){
		return numPages;
	}
	public Author getAuthor(){
		return author;
	}
	public double getPrice(){
		return price;
	}
	public void setPrice(double price){
		this.price = price;
	}
	public String toString(){
		return title +" written by "+ author.getAuthorName()+ " ("+numPages+" pages)" +"$"+price;
	}
}
```

Example



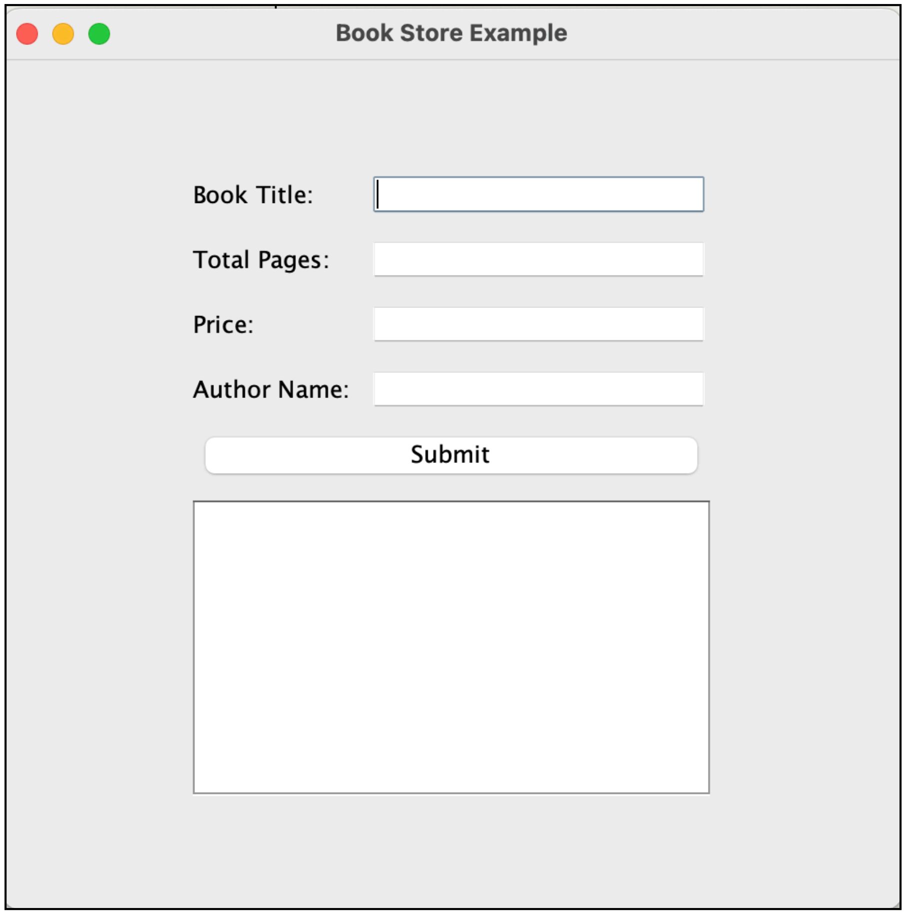
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class BookStoreGUI extends JFrame implements ActionListener {
    private JLabel bookTitleLabel, pagesLabel, priceLabel, authorLabel;
    private JTextField titleField, pageNumField, priceField, authorField;
    private JButton submitButton;
    private JTextArea textArea;

    public BookStoreGUI() {
        setTitle("Book Store Example");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new GridBagLayout());

```

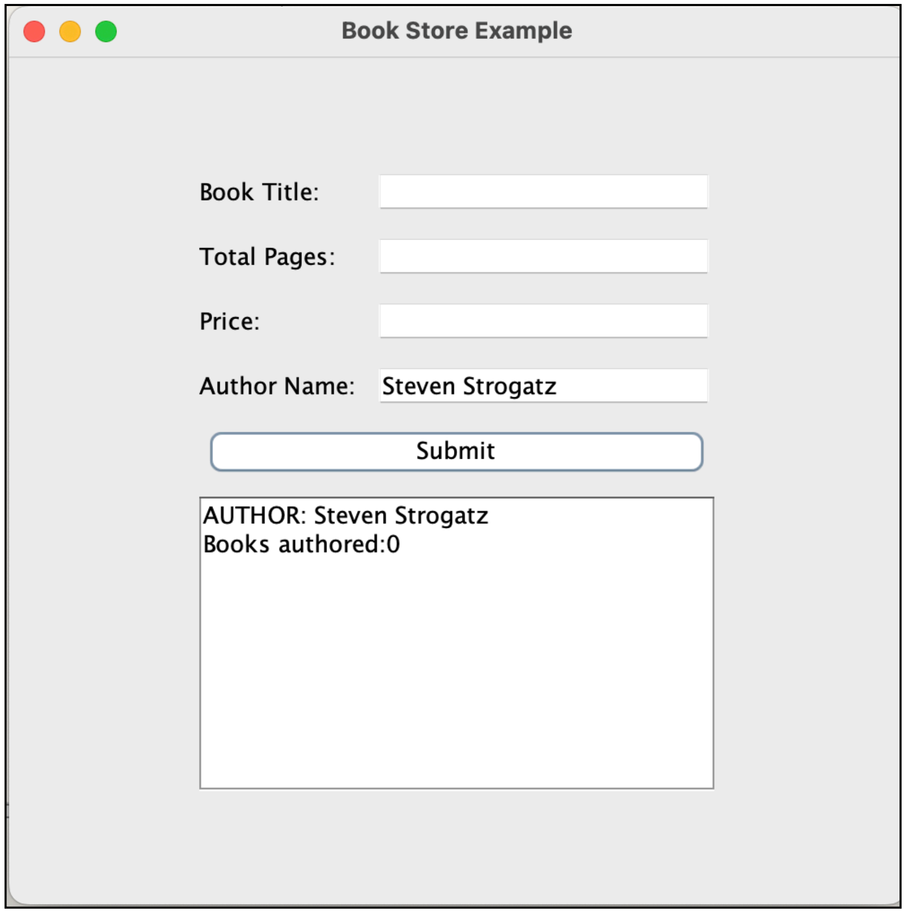
.
.
.



```

    public void actionPerformed(ActionEvent e) {
        if (e.getSource().equals(submitButton)) {
            if(!authorField.getText().equals("")){
                Author a = new Author (authorField.getText());

                textArea.append(a.toString() +"\n");
            }
        }
    }
```

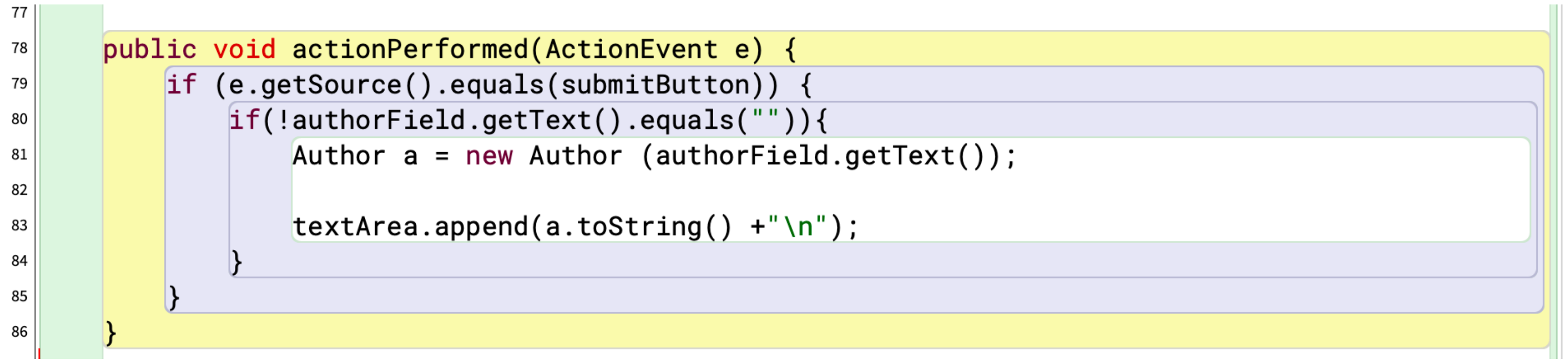
Example





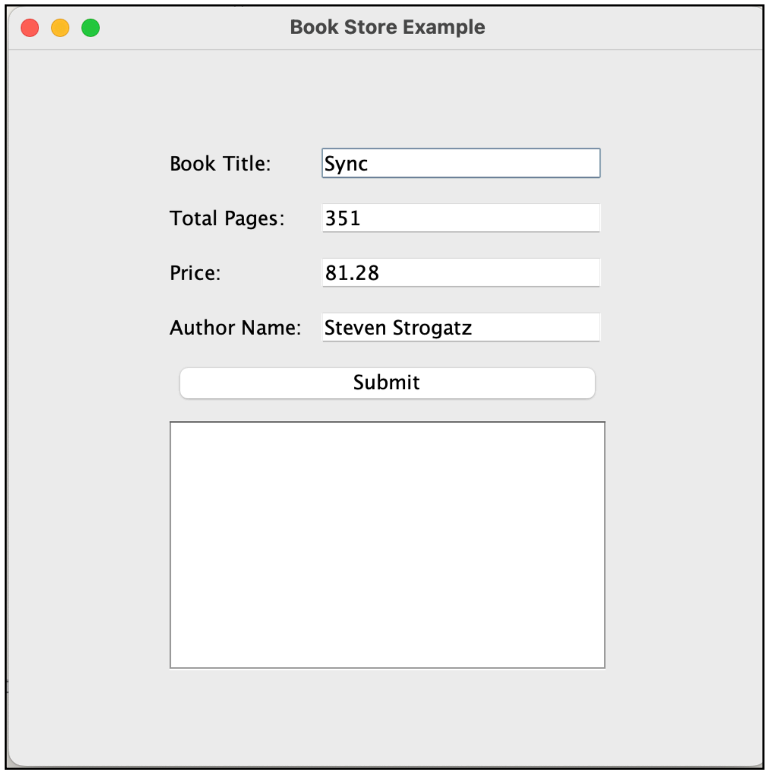
Steven Strogatz

```
AUTHOR: Steven Strogatz
Books authored:0
```



```java
public void actionPerformed(ActionEvent e) {
    if (e.getSource().equals(submitButton)) {
        if(!authorField.getText().equals("")){
            Author a = new Author (authorField.getText());

            textArea.append(a.toString() +"\n");
        }
    }
}
```

Example

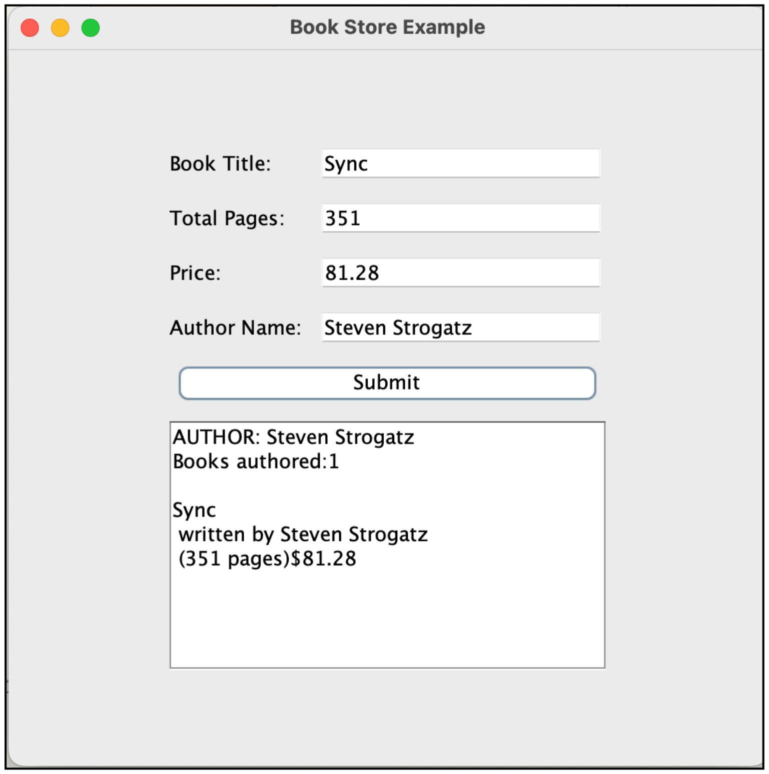


Book Title: Sync

Total Pages: 351

Price: 81.28

Author Name: Steven Strogatz



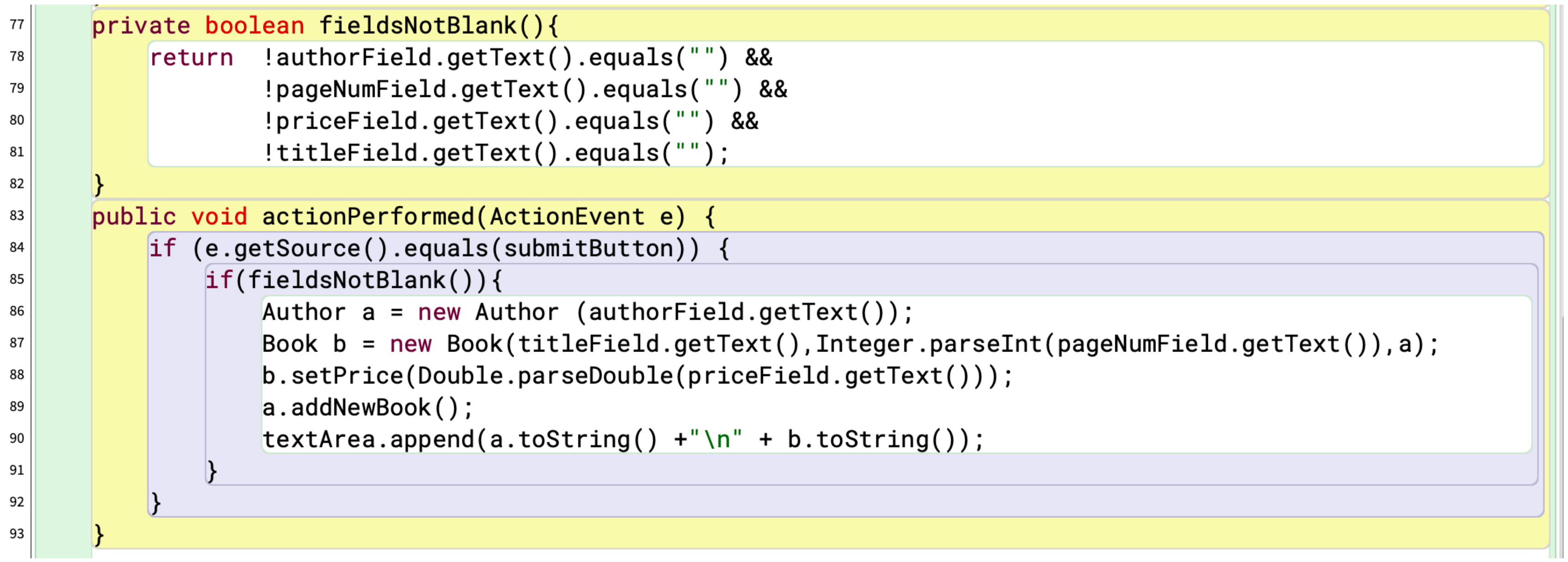
Book Title: Sync

Total Pages: 351

Price: 81.28

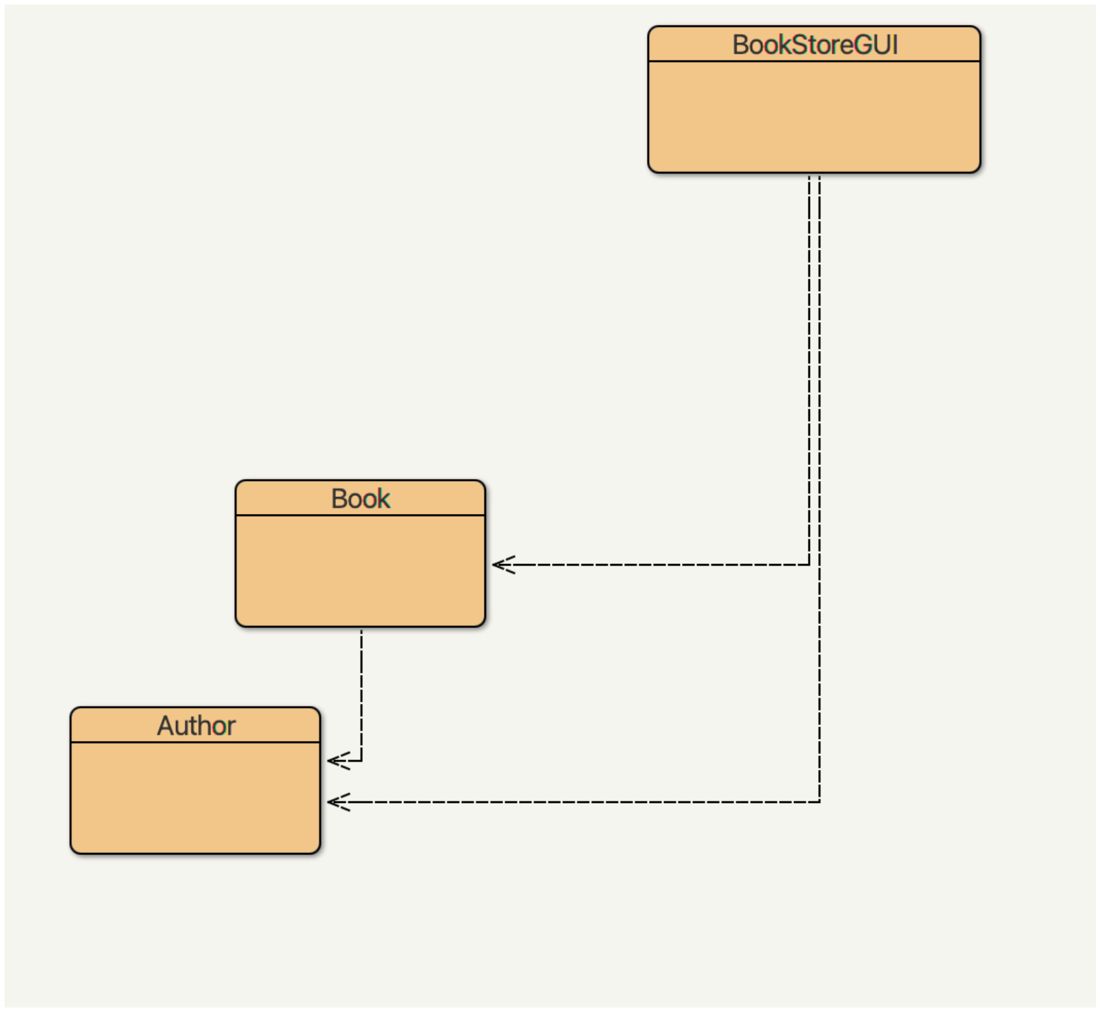
Author Name: Steven Strogatz

```
AUTHOR: Steven Strogatz
Books authored:1

Sync
 written by Steven Strogatz
 (351 pages)$81.28
```

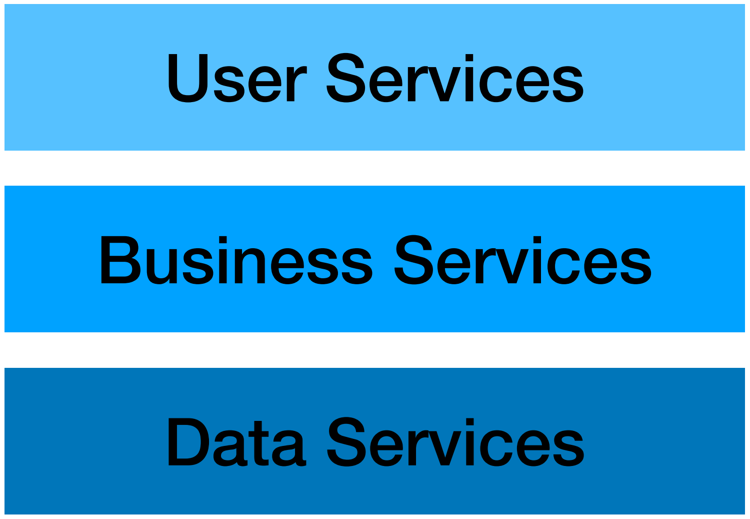


```java
private boolean fieldsNotBlank(){
    return  !authorField.getText().equals("") &&
            !pageNumField.getText().equals("") &&
            !priceField.getText().equals("") &&
            !titleField.getText().equals("");
}
public void actionPerformed(ActionEvent e) {
    if (e.getSource().equals(submitButton)) {
        if(fieldsNotBlank()){
            Author a = new Author (authorField.getText());
            Book b = new Book(titleField.getText(),Integer.parseInt(pageNumField.getText()),a);
            b.setPrice(Double.parseDouble(priceField.getText()));
            a.addNewBook();
            textArea.append(a.toString() +"\n" + b.toString());
        }
    }
}
```

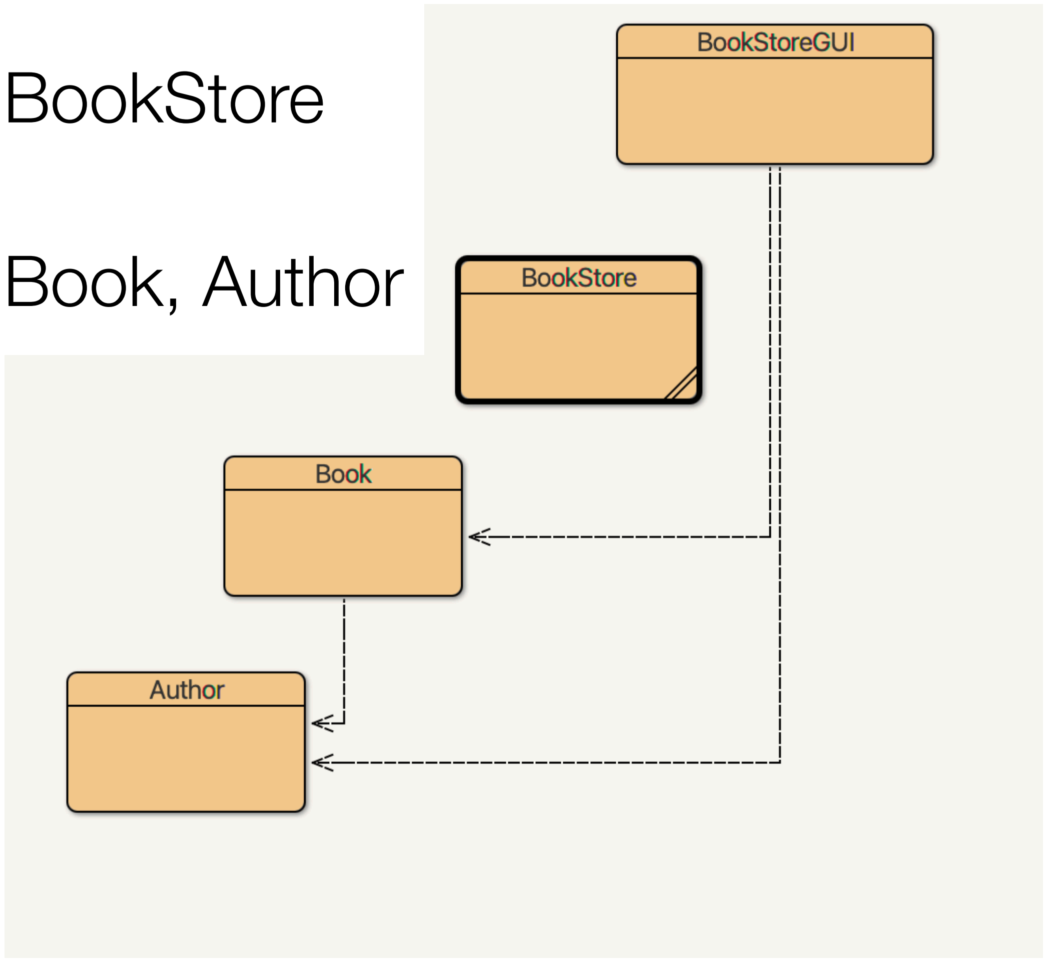
Example - Tight Coupling



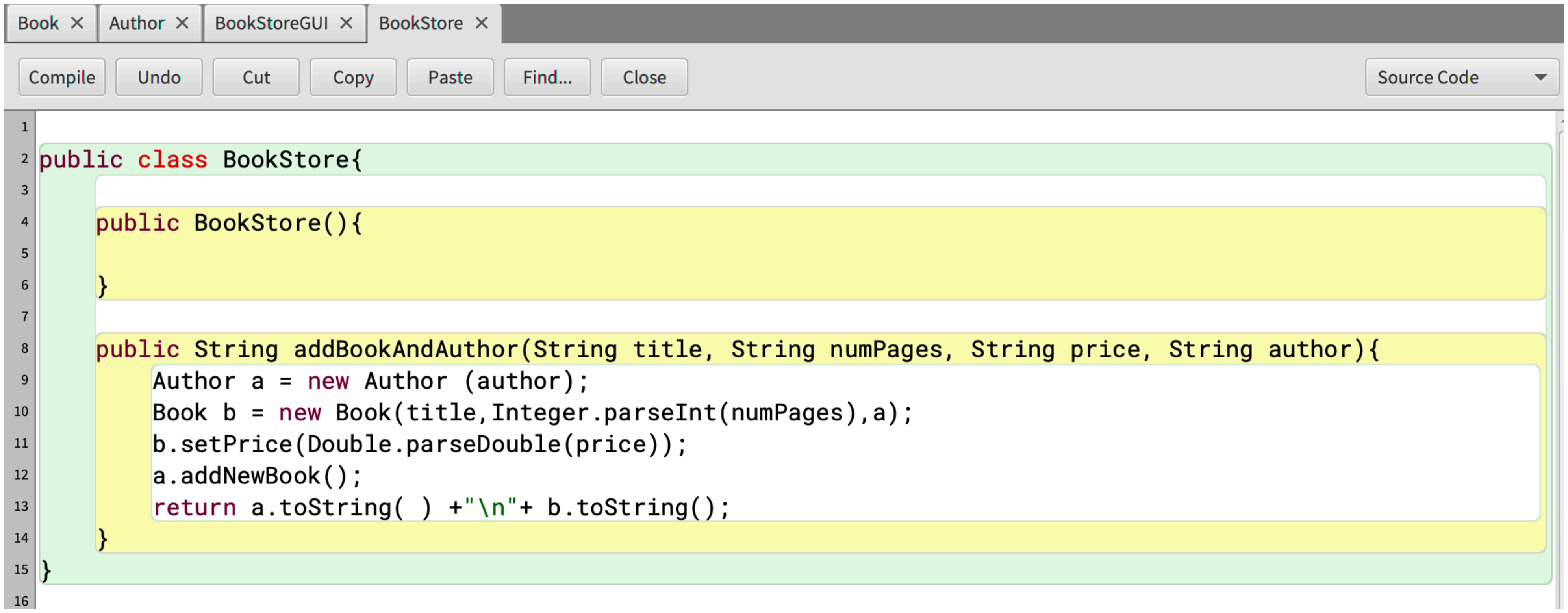
Applying Model View Separation



- BookStoreGUI
- BookStore
- Book, Author

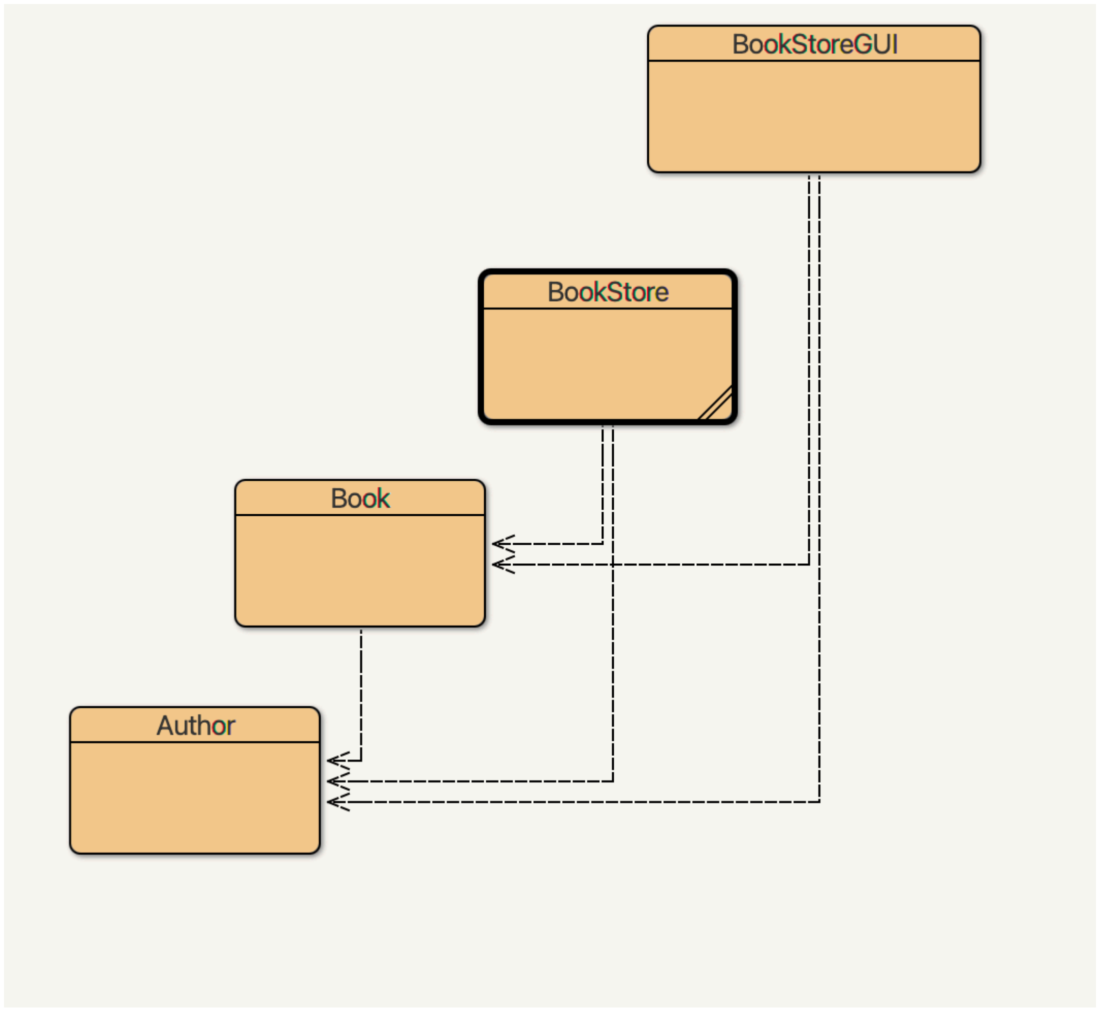


Applying Model View Separation

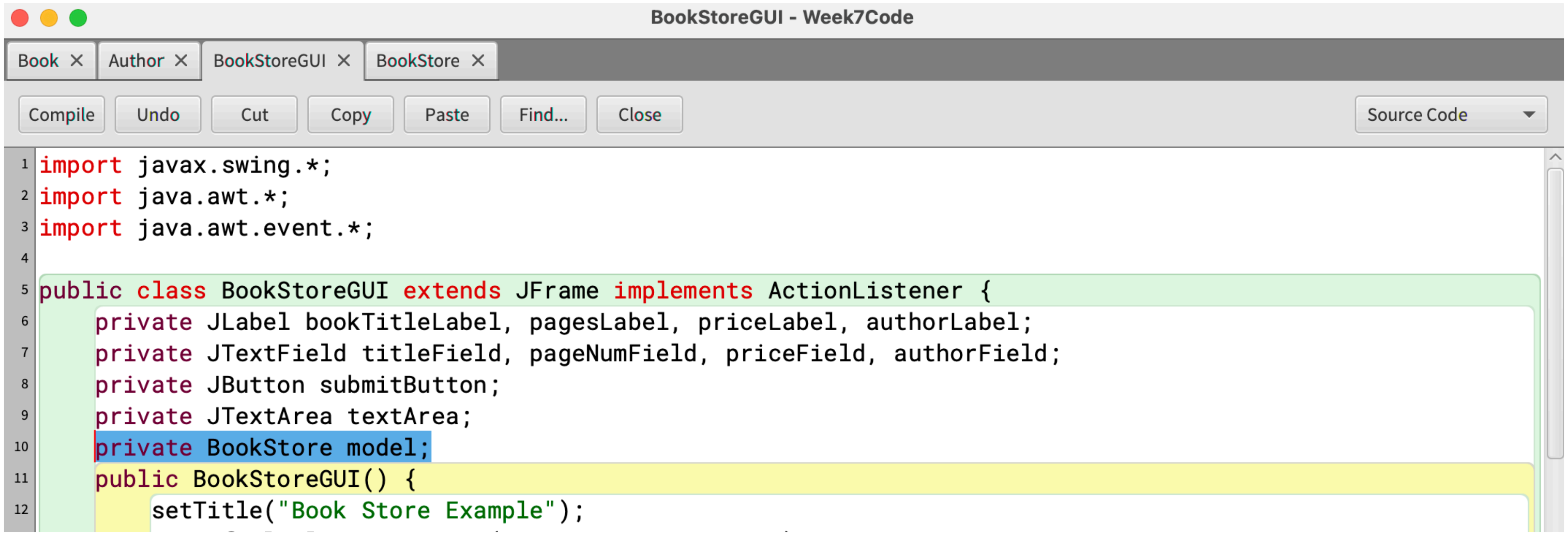


```

public class BookStore{

	public BookStore(){

	}

	public String addBookAndAuthor(String title, String numPages, String price, String author){
		Author a = new Author (author);
		Book b = new Book(title,Integer.parseInt(numPages),a);
		b.setPrice(Double.parseDouble(price));
		a.addNewBook();
		return a.toString( ) +"\n"+ b.toString();
	}
}

```

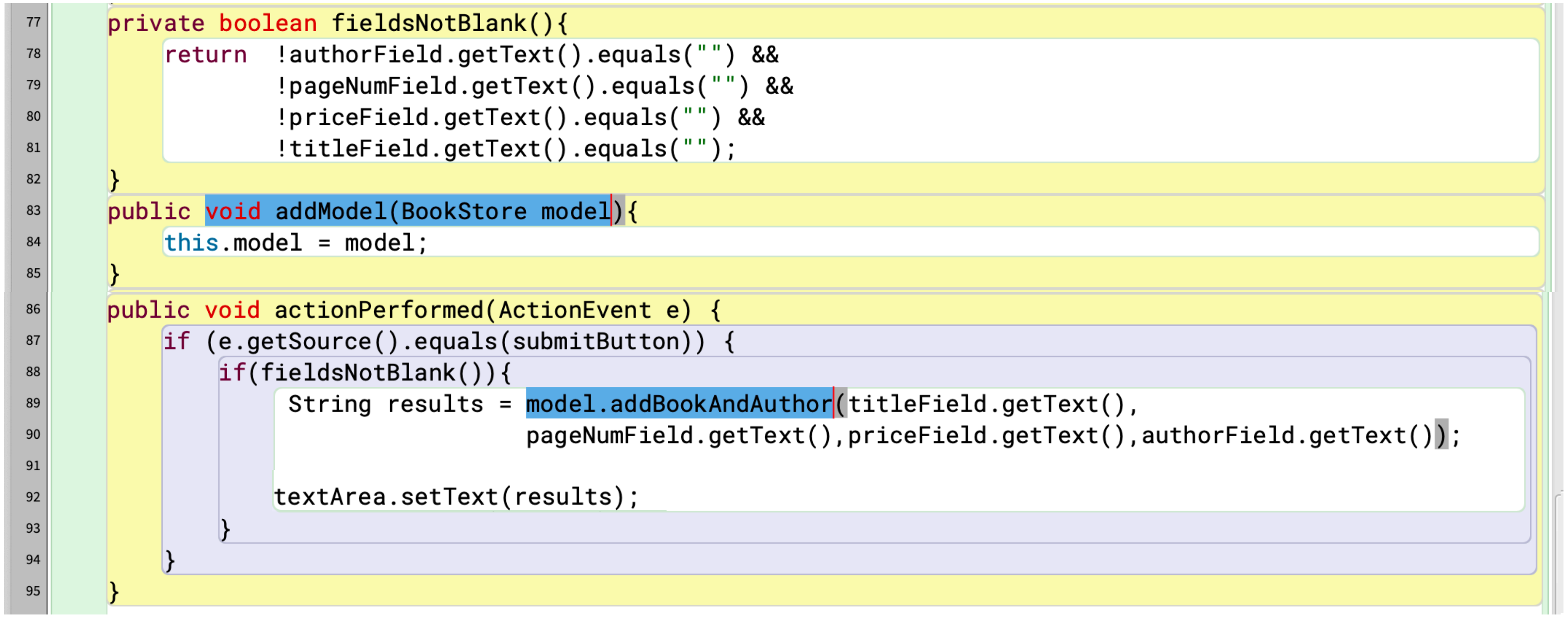
Decoupling of View to Data Layers



Applying Model View Separation

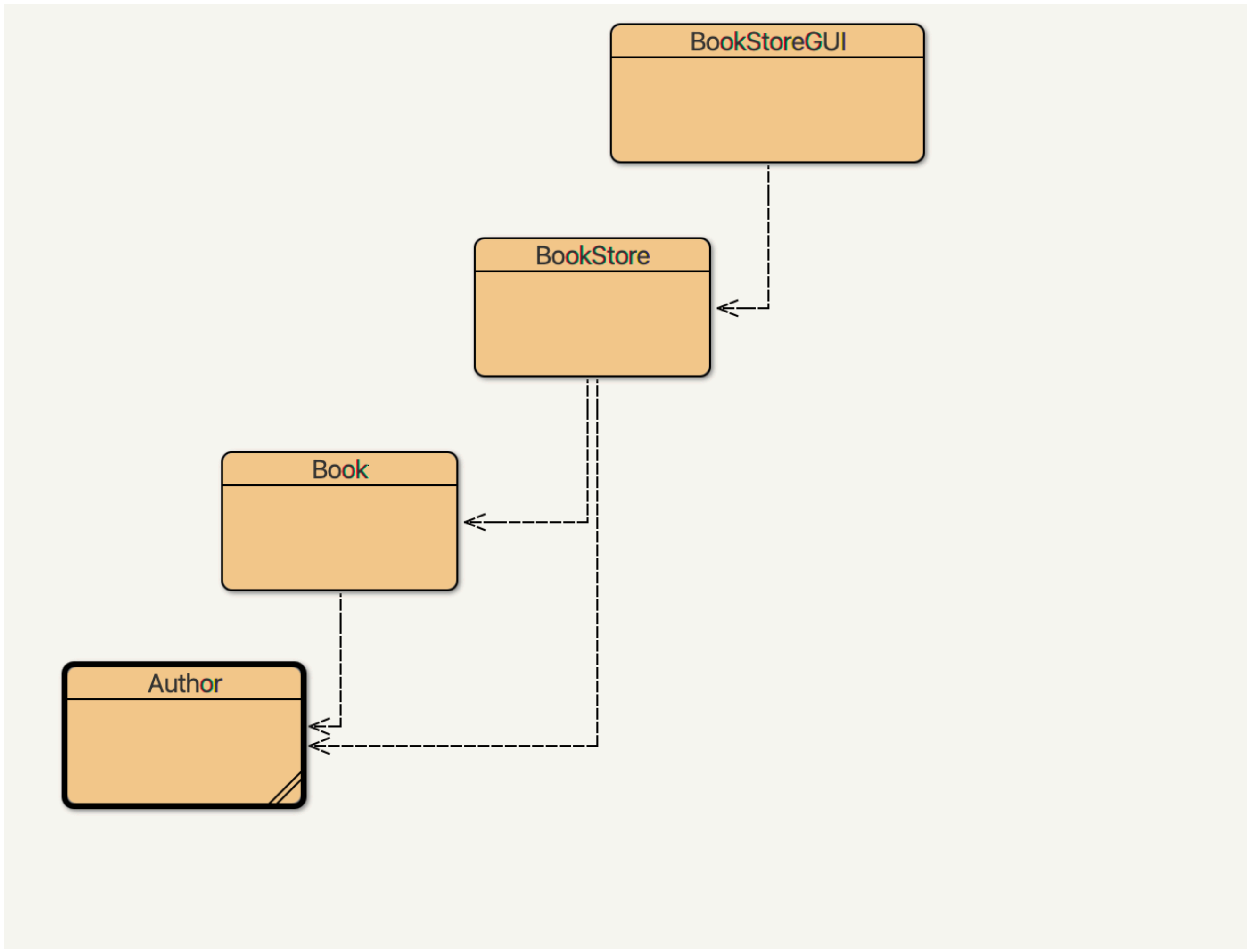

BookStoreGUI - Week7Code
Book ✕ Author ✕ BookStoreGUI ✕ BookStore ✕
Compile Undo Cut Copy Paste Find... Close
Source Code

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class BookStoreGUI extends JFrame implements ActionListener {
    private JLabel bookTitleLabel, pagesLabel, priceLabel, authorLabel;
    private JTextField titleField, pageNumField, priceField, authorField;
    private JButton submitButton;
    private JTextArea textArea;
    private BookStore model;
    public BookStoreGUI() {
        setTitle("Book Store Example");
```



```
    private boolean fieldsNotBlank(){
        return  !authorField.getText().equals("") &&
                !pageNumField.getText().equals("") &&
                !priceField.getText().equals("") &&
                !titleField.getText().equals("");
    }
    public void addModel(BookStore model){
        this.model = model;
    }
    public void actionPerformed(ActionEvent e) {
        if (e.getSource().equals(submitButton)) {
            if(fieldsNotBlank()){
                String results = model.addBookAndAuthor(titleField.getText(),
                                 pageNumField.getText(),priceField.getText(),authorField.getText());

               textArea.setText(results);
            }
        }
    }
```

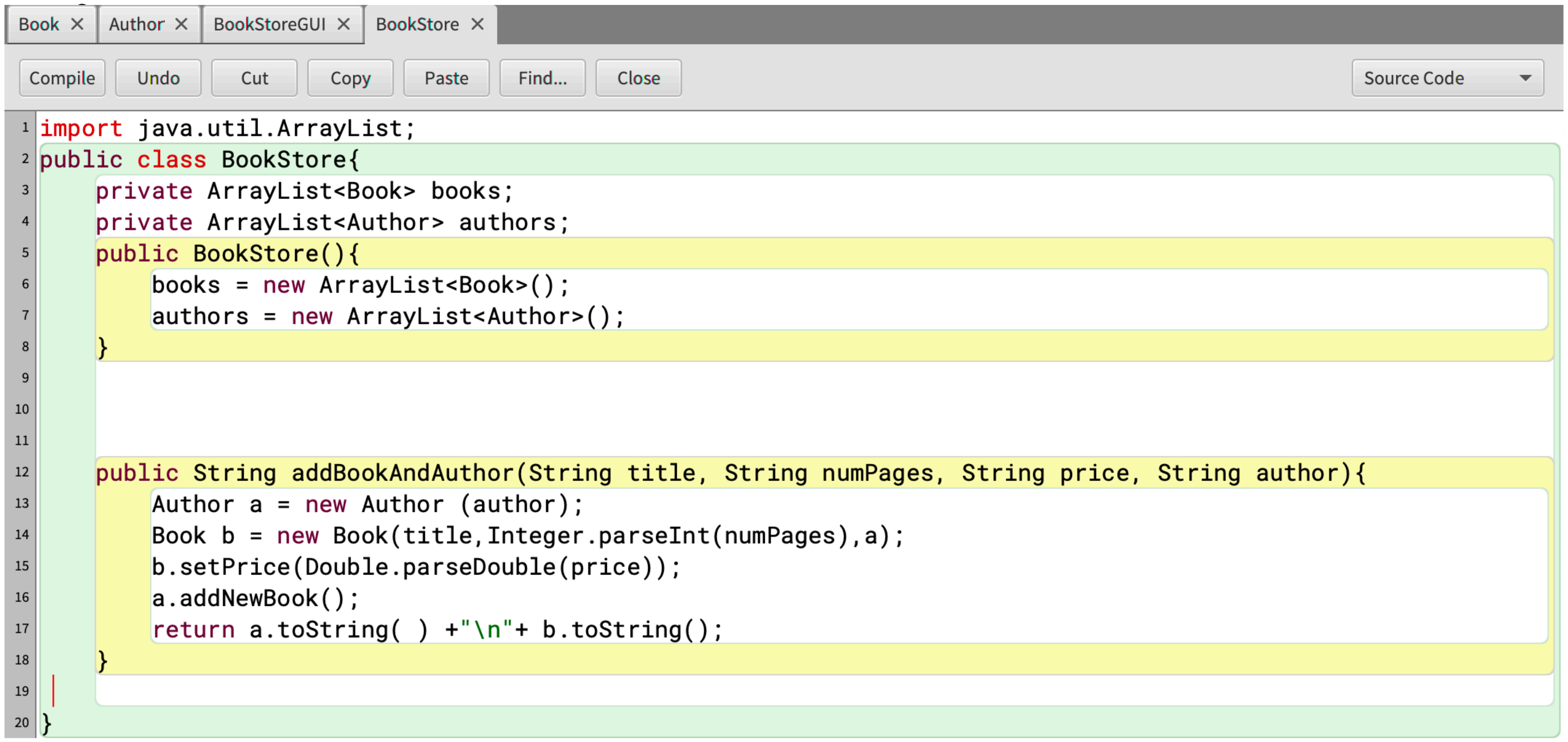
Applying Model View Separation



Applying Model View Separation

Tweaks

- Add equals(..) to Book and Author



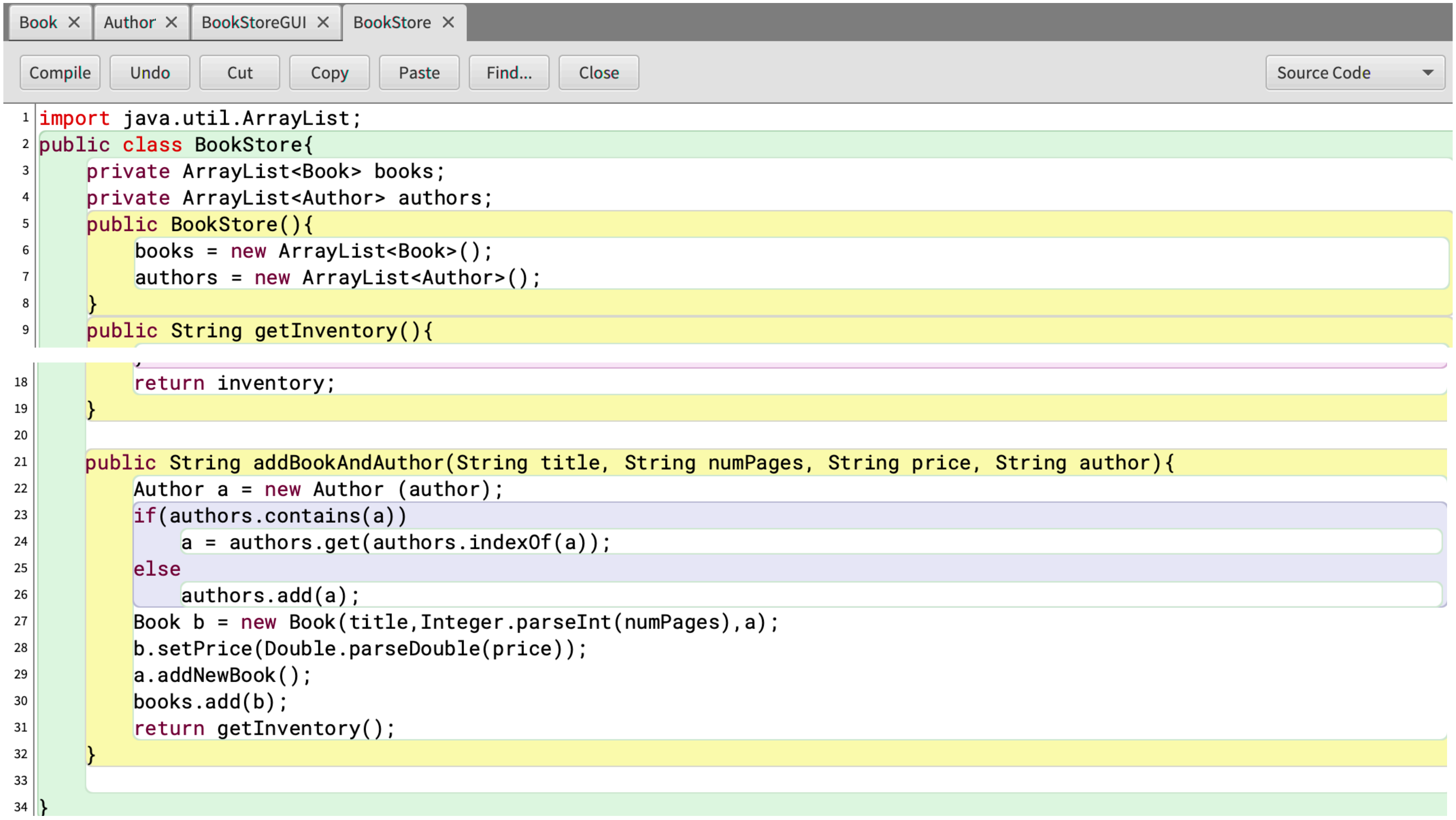
Book ✕ Author ✕ BookStoreGUI ✕ BookStore ✕
Compile Undo Cut Copy Paste Find... Close
Source Code

```java
import java.util.ArrayList;
public class BookStore{
    private ArrayList<Book> books;
    private ArrayList<Author> authors;
    public BookStore(){
        books = new ArrayList<Book>();
        authors = new ArrayList<Author>();
    }



    public String addBookAndAuthor(String title, String numPages, String price, String author){
        Author a = new Author (author);
        Book b = new Book(title,Integer.parseInt(numPages),a);
        b.setPrice(Double.parseDouble(price));
        a.addNewBook();
        return a.toString( ) +"\n"+ b.toString();
    }

}
```

Applying Model View Separation

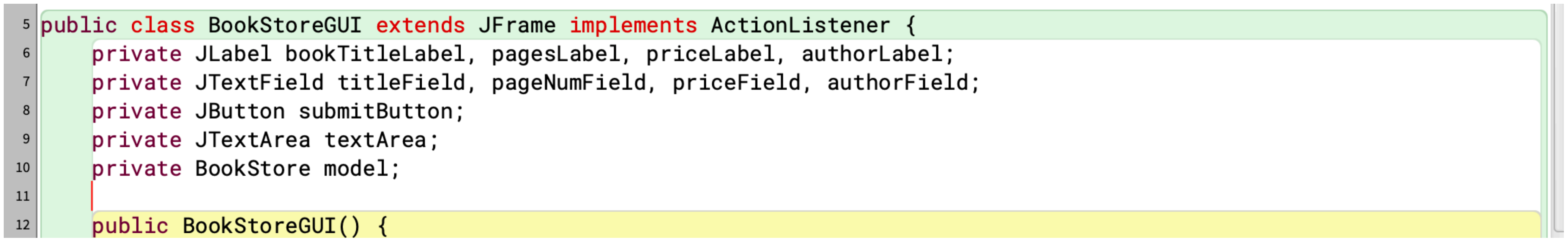
Tweaks

- Add equals(..) to Book and Author

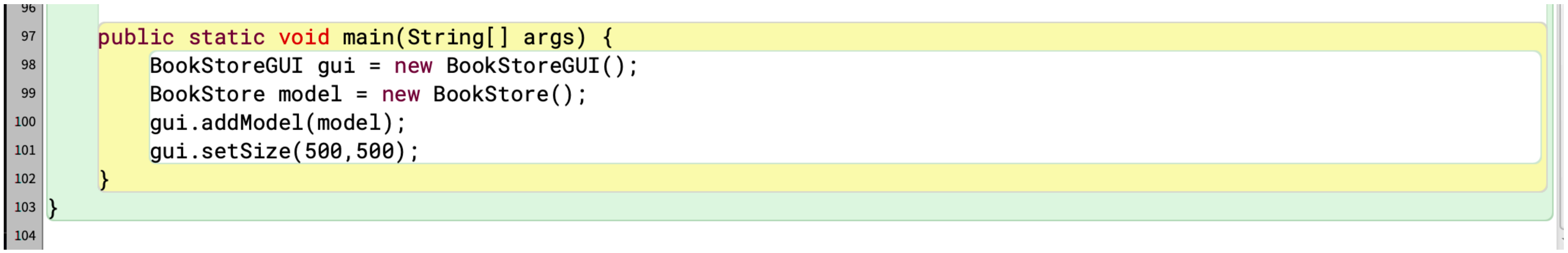


```java
import java.util.ArrayList;
public class BookStore{
    private ArrayList<Book> books;
    private ArrayList<Author> authors;
    public BookStore(){
        books = new ArrayList<Book>();
        authors = new ArrayList<Author>();
    }
    public String getInventory(){
        }
        return inventory;
    }

    public String addBookAndAuthor(String title, String numPages, String price, String author){
        Author a = new Author (author);
        if(authors.contains(a))
            a = authors.get(authors.indexOf(a));
        else
            authors.add(a);
        Book b = new Book(title,Integer.parseInt(numPages),a);
        b.setPrice(Double.parseDouble(price));
        a.addNewBook();
        books.add(b);
        return getInventory();
    }

}
```

Applying Model View Separation

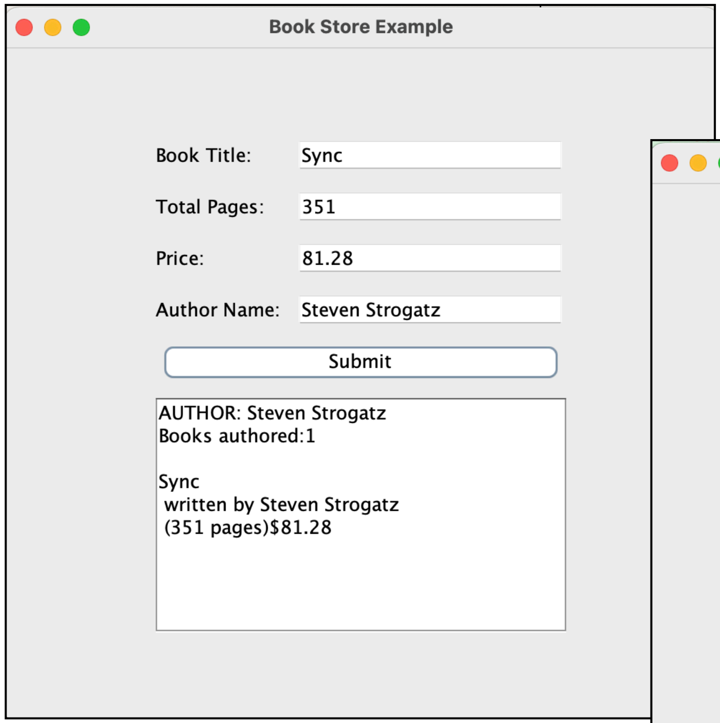


```java
public class BookStoreGUI extends JFrame implements ActionListener {
    private JLabel bookTitleLabel, pagesLabel, priceLabel, authorLabel;
    private JTextField titleField, pageNumField, priceField, authorField;
    private JButton submitButton;
    private JTextArea textArea;
    private BookStore model;

    public BookStoreGUI() {
```



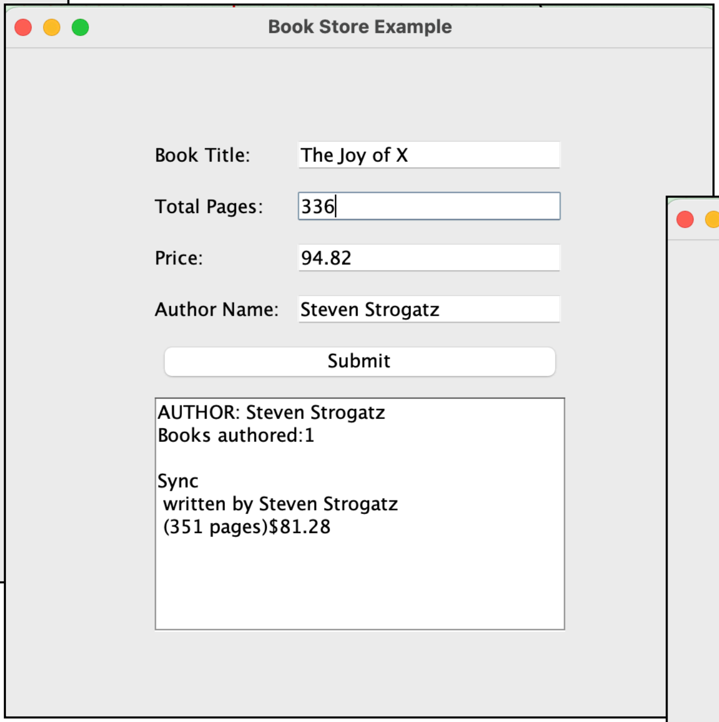
```java

    public static void main(String[] args) {
        BookStoreGUI gui = new BookStoreGUI();
        BookStore model = new BookStore();
        gui.addModel(model);
        gui.setSize(500,500);
    }
}

```

Applying Model View Separation



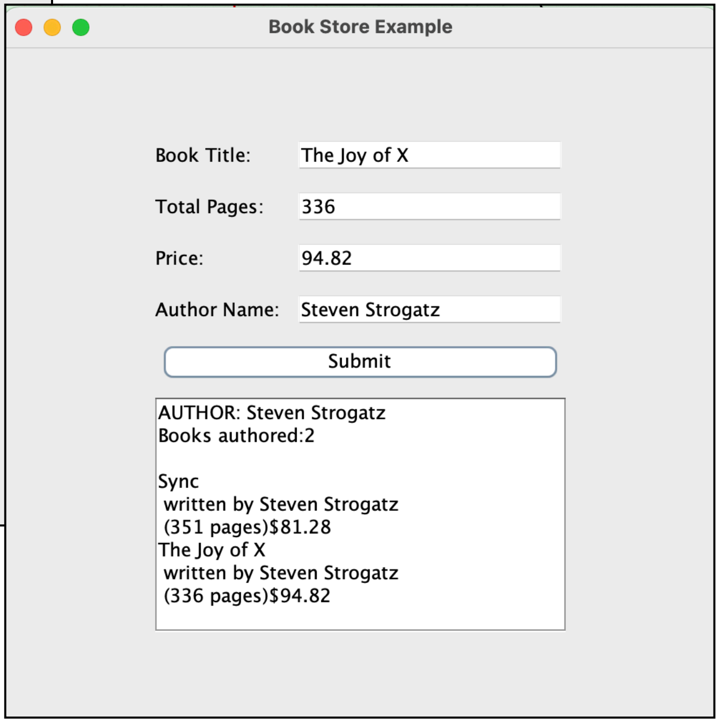
Sync
351
81.28
Steven Strogatz

```
AUTHOR: Steven Strogatz
Books authored:1

Sync
 written by Steven Strogatz
 (351 pages)$81.28
```



The Joy of X
336
94.82
Steven Strogatz

```
AUTHOR: Steven Strogatz
Books authored:1

Sync
 written by Steven Strogatz
 (351 pages)$81.28
```



The Joy of X
336
94.82
Steven Strogatz

```
AUTHOR: Steven Strogatz
Books authored:2

Sync
 written by Steven Strogatz
 (351 pages)$81.28
The Joy of X
 written by Steven Strogatz
 (336 pages)$94.82
```

Summary

Today you learned about:

- Model View Separation (definition, code example)
- Coupling (definition, code example)
- Cohesion (definition, code example)