

# 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.

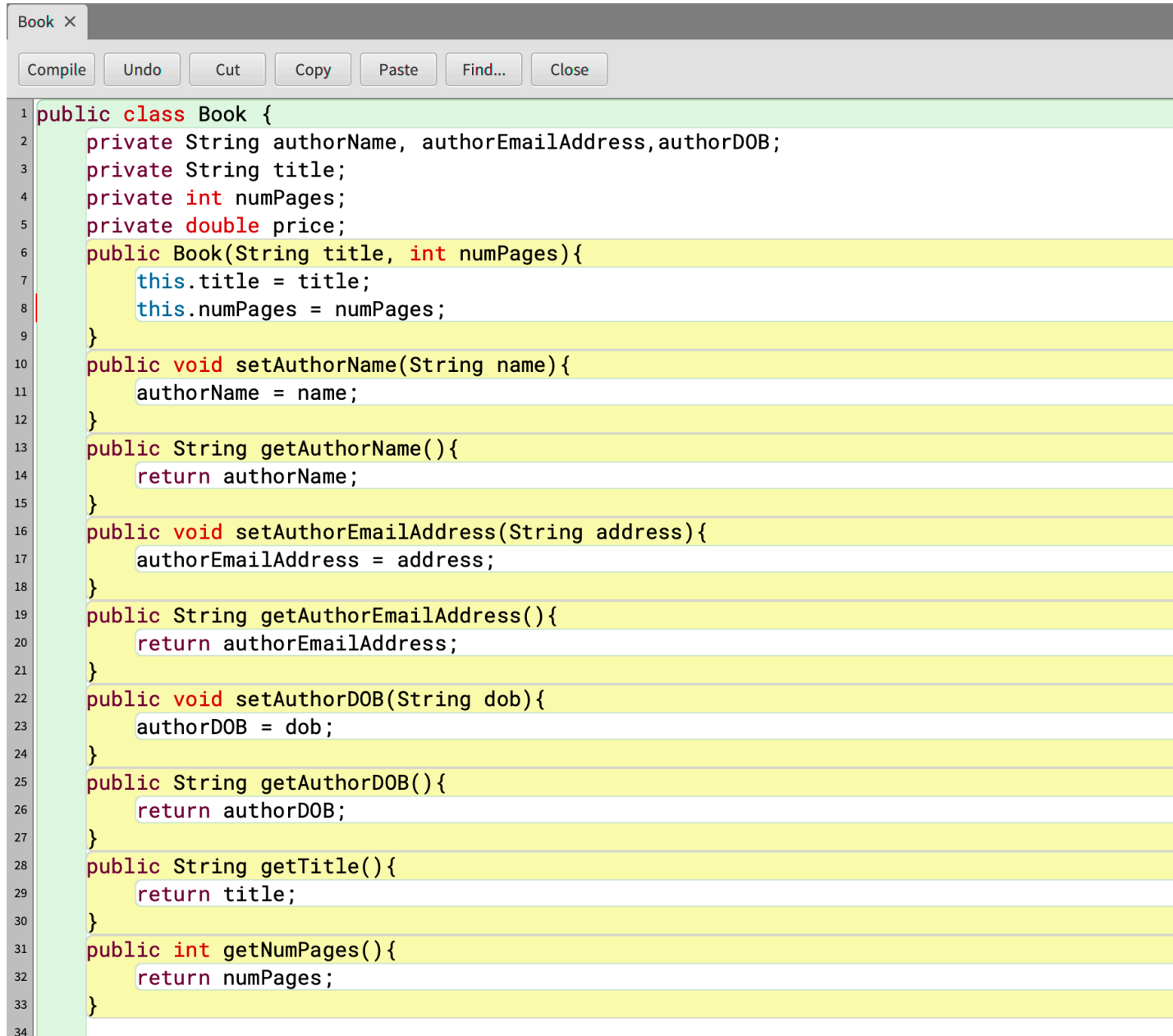
# Weak Cohesion

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

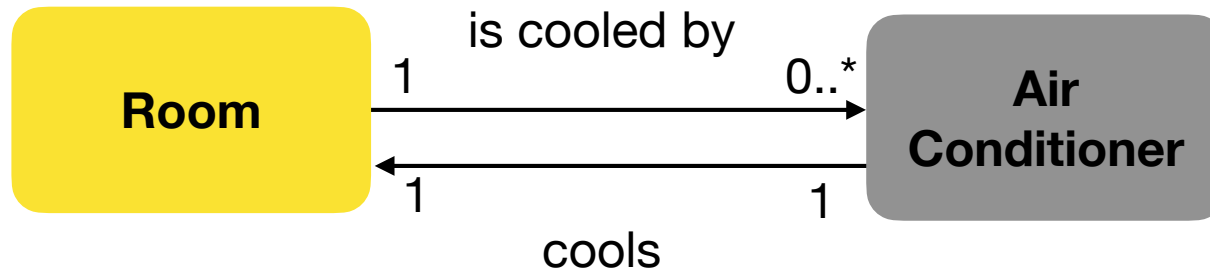


```
1 public class Book {
2     private String authorName, authorEmailAddress, authorDOB;
3     private String title;
4     private int numPages;
5     private double price;
6     public Book(String title, int numPages){
7         this.title = title;
8         this.numPages = numPages;
9     }
10    public void setAuthorName(String name){
11        authorName = name;
12    }
13    public String getAuthorName(){
14        return authorName;
15    }
16    public void setAuthorEmailAddress(String address){
17        authorEmailAddress = address;
18    }
19    public String getAuthorEmailAddress(){
20        return authorEmailAddress;
21    }
22    public void setAuthorDOB(String dob){
23        authorDOB = dob;
24    }
25    public String getAuthorDOB(){
26        return authorDOB;
27    }
28    public String getTitle(){
29        return title;
30    }
31    public int getNumPages(){
32        return numPages;
33    }
34 }
```

# 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

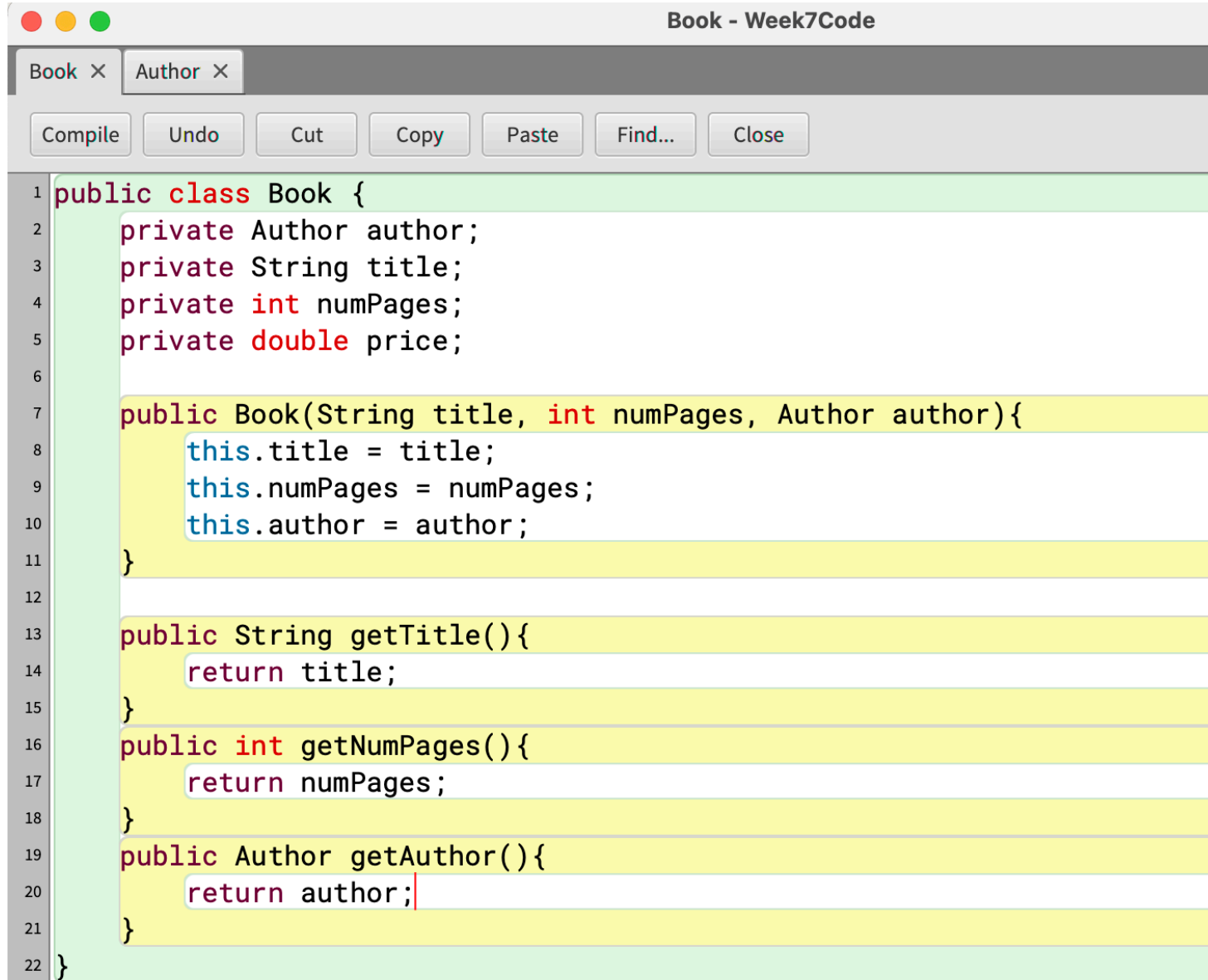
Object-Oriented coupling refers to the degree or strength of interconnection among classes.

**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.

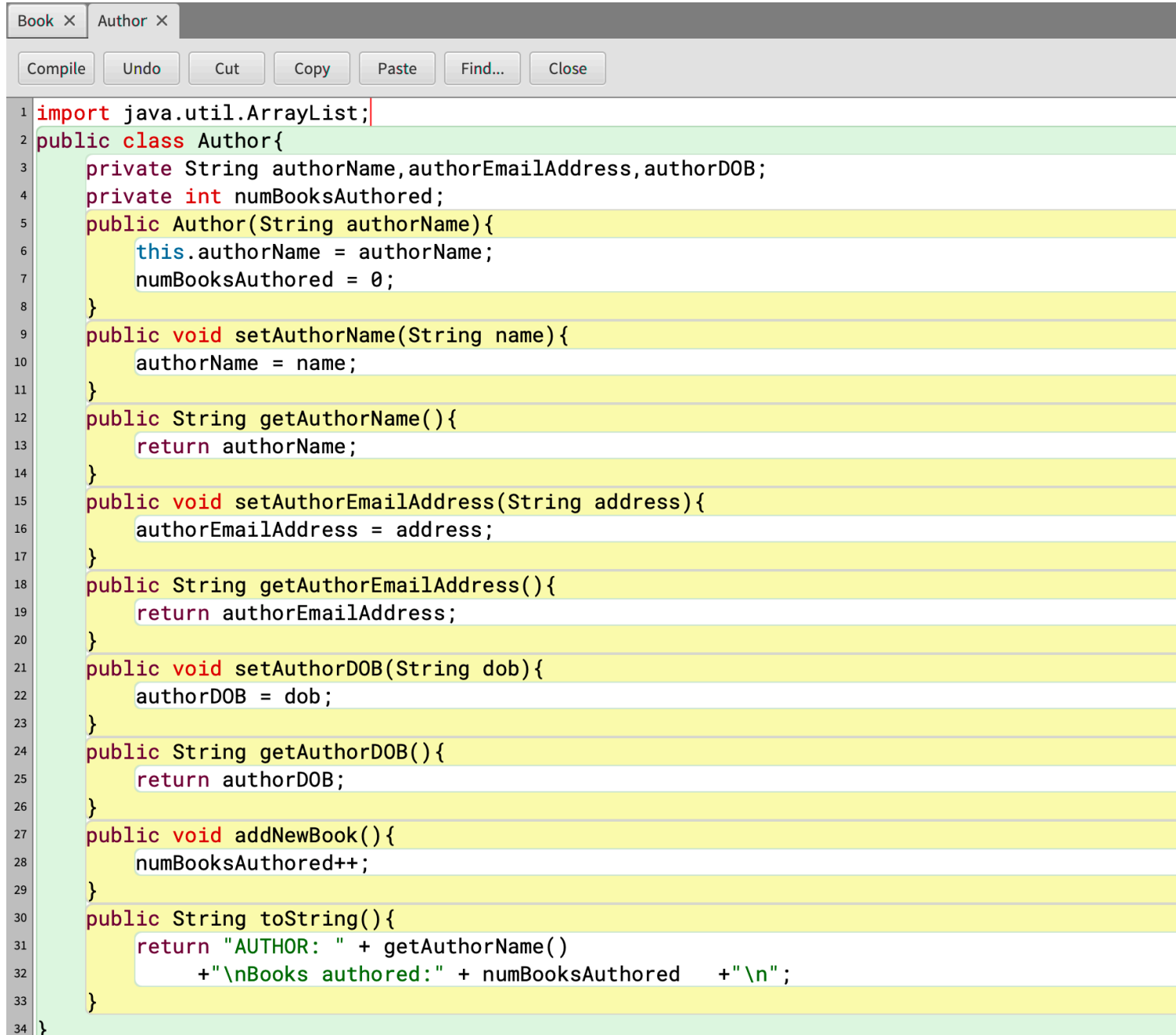
# Example - Book.java



```
1 public class Book {
2     private Author author;
3     private String title;
4     private int numPages;
5     private double price;
6
7     public Book(String title, int numPages, Author author){
8         this.title = title;
9         this.numPages = numPages;
10        this.author = author;
11    }
12
13    public String getTitle(){
14        return title;
15    }
16    public int getNumPages(){
17        return numPages;
18    }
19    public Author getAuthor(){
20        return author;
21    }
22 }
```

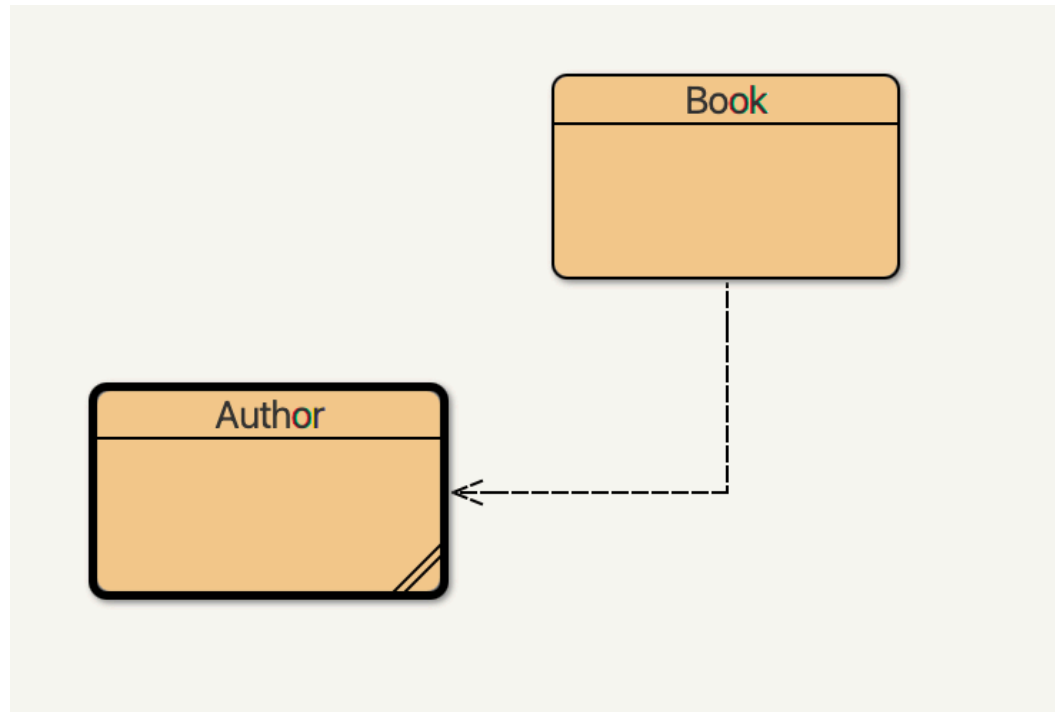


# Example - Author.java

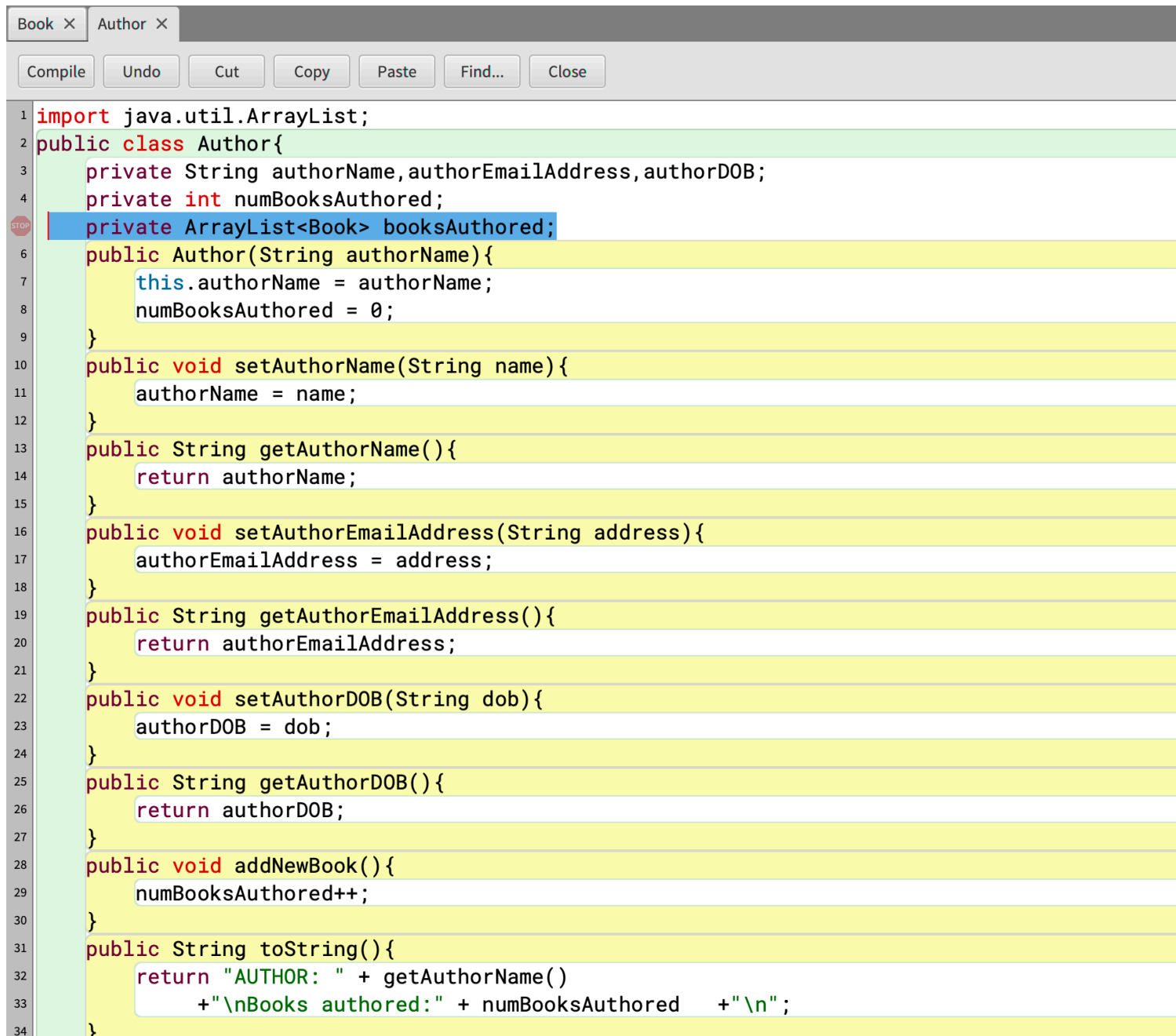


```
1 import java.util.ArrayList;
2 public class Author{
3     private String authorName,authorEmailAddress,authorDOB;
4     private int numBooksAuthored;
5     public Author(String authorName){
6         this.authorName = authorName;
7         numBooksAuthored = 0;
8     }
9     public void setAuthorName(String name){
10         authorName = name;
11     }
12     public String getAuthorName(){
13         return authorName;
14     }
15     public void setAuthorEmailAddress(String address){
16         authorEmailAddress = address;
17     }
18     public String getAuthorEmailAddress(){
19         return authorEmailAddress;
20     }
21     public void setAuthorDOB(String dob){
22         authorDOB = dob;
23     }
24     public String getAuthorDOB(){
25         return authorDOB;
26     }
27     public void addNewBook(){
28         numBooksAuthored++;
29     }
30     public String toString(){
31         return "AUTHOR: " + getAuthorName()
32             + "\nBooks authored:" + numBooksAuthored + "\n";
33     }
34 }
```

# Example - Loose Coupling

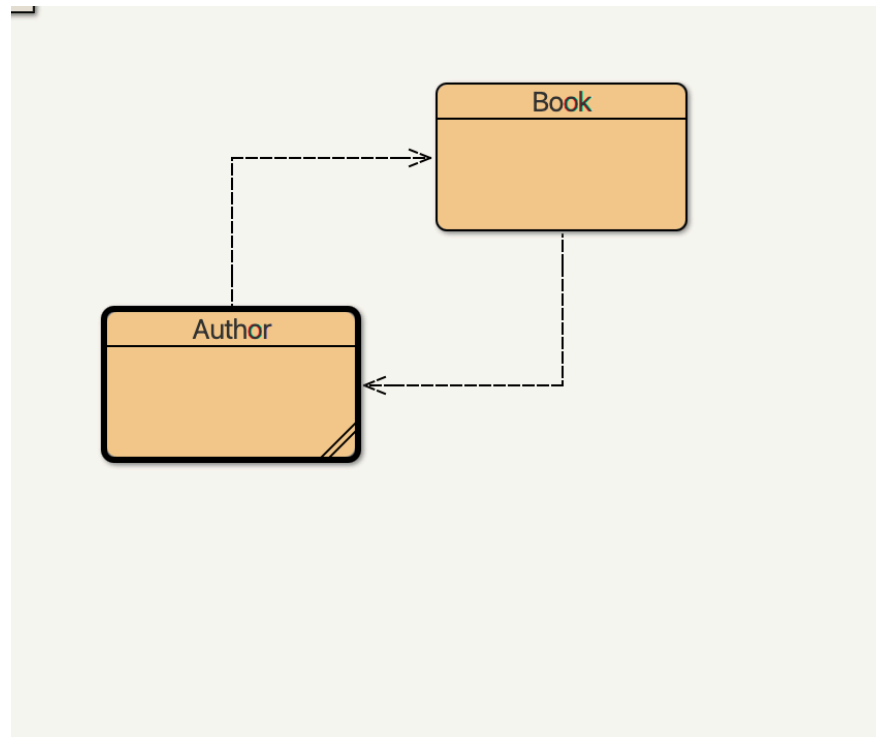


# Example - Author.java



```
1 import java.util.ArrayList;
2 public class Author{
3     private String authorName,authorEmailAddress,authorDOB;
4     private int numBooksAuthored;
5     private ArrayList<Book> booksAuthored;
6     public Author(String authorName){
7         this.authorName = authorName;
8         numBooksAuthored = 0;
9     }
10    public void setAuthorName(String name){
11        authorName = name;
12    }
13    public String getAuthorName(){
14        return authorName;
15    }
16    public void setAuthorEmailAddress(String address){
17        authorEmailAddress = address;
18    }
19    public String getAuthorEmailAddress(){
20        return authorEmailAddress;
21    }
22    public void setAuthorDOB(String dob){
23        authorDOB = dob;
24    }
25    public String getAuthorDOB(){
26        return authorDOB;
27    }
28    public void addNewBook(){
29        numBooksAuthored++;
30    }
31    public String toString(){
32        return "AUTHOR: " + getAuthorName()
33            + "\nBooks authored:" + numBooksAuthored + "\n";
34    }
```

# 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



The diagram illustrates the three layers of a Three-Tier Architecture. It consists of three stacked rectangular boxes. The top box is light blue and labeled 'User Services'. The middle box is a medium blue and labeled 'Business Services'. The bottom box is a dark blue and labeled 'Data Services'. The boxes are separated by thin white gaps.

**User Services**

**Business Services**

**Data Services**



# Model View Separation

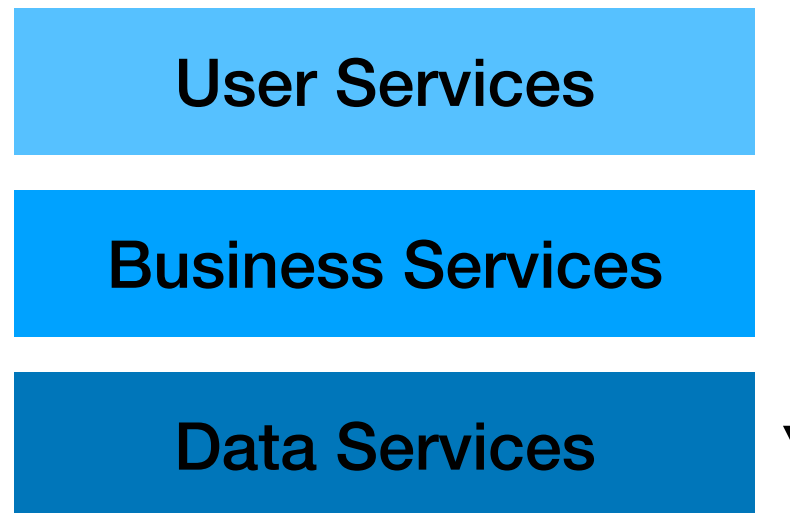
The Model-View Separation design pattern states that model object (domain objects or objects in the Business Services tier) should not have direct knowledge of or be directly coupled to view objects (user interface objects).

# 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**.

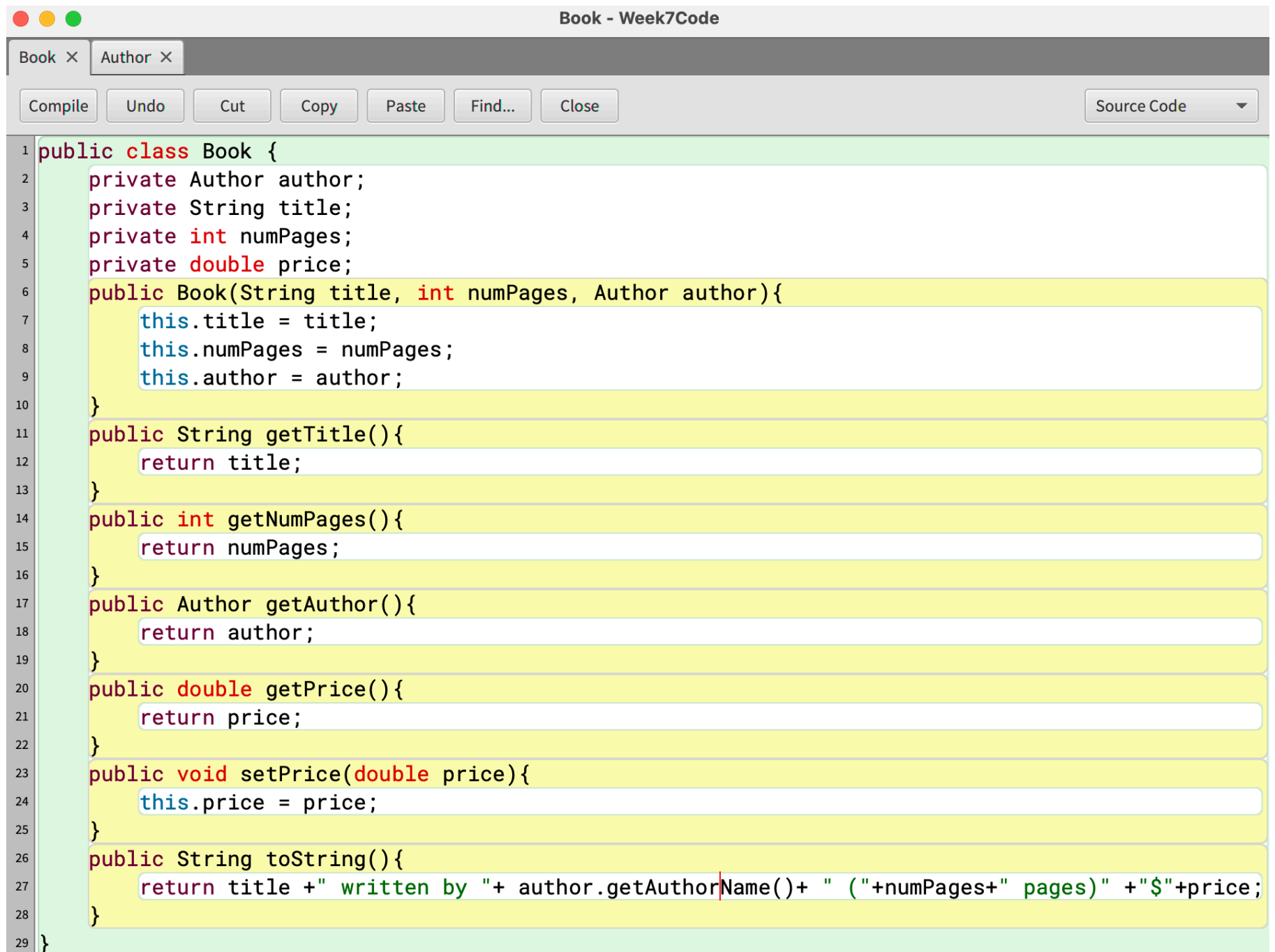


# Advantages of Model-View Separation

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

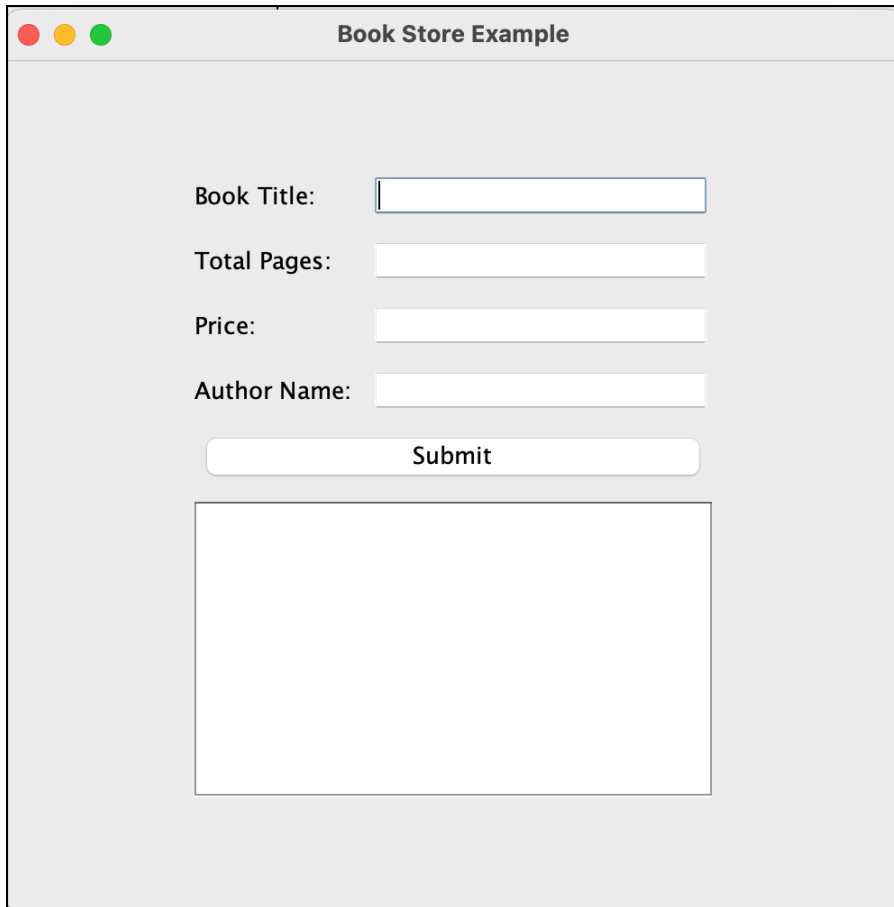


```
1 public class Book {
2     private Author author;
3     private String title;
4     private int numPages;
5     private double price;
6     public Book(String title, int numPages, Author author){
7         this.title = title;
8         this.numPages = numPages;
9         this.author = author;
10    }
11    public String getTitle(){
12        return title;
13    }
14    public int getNumPages(){
15        return numPages;
16    }
17    public Author getAuthor(){
18        return author;
19    }
20    public double getPrice(){
21        return price;
22    }
23    public void setPrice(double price){
24        this.price = price;
25    }
26    public String toString(){
27        return title + " written by " + author.getAuthorName() + " (" + numPages + " pages)" + "$" + price;
28    }
29 }
```

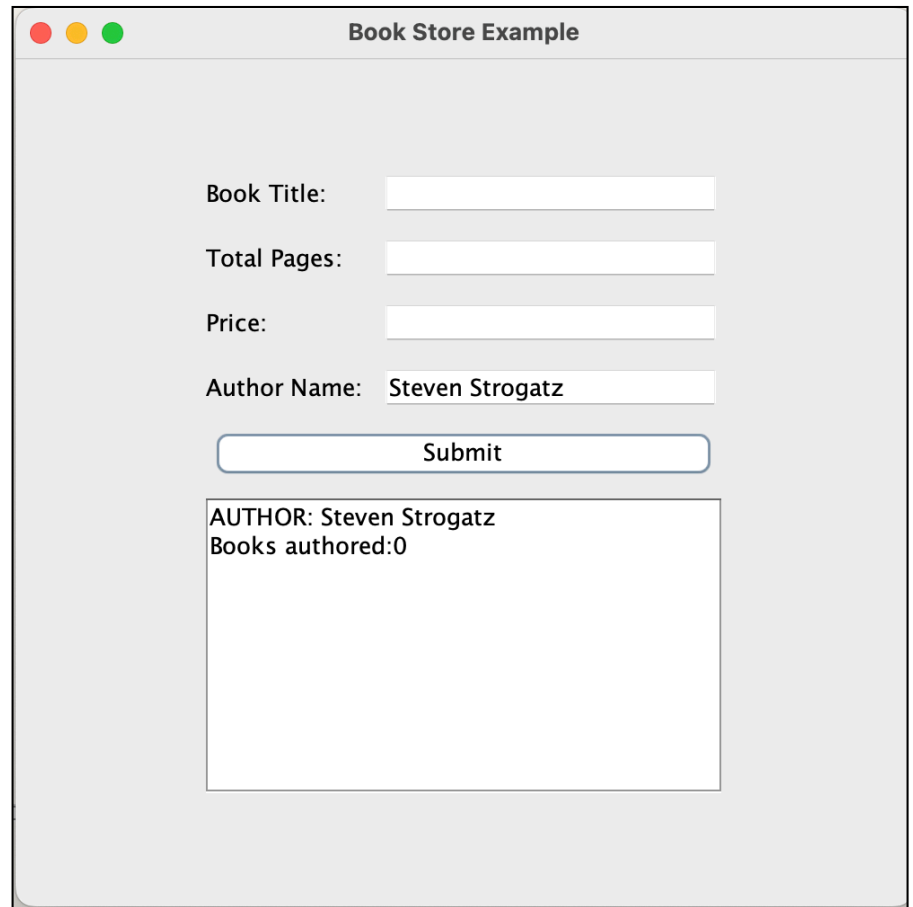
# Example

```
1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.*;
4
5 public class BookStoreGUI extends JFrame implements ActionListener {
6     private JLabel bookTitleLabel, pagesLabel, priceLabel, authorLabel;
7     private JTextField titleField, pageNumField, priceField, authorField;
8     private JButton submitButton;
9     private JTextArea textArea;
10
11     public BookStoreGUI() {
12         setTitle("Book Store Example");
13         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14         setLayout(new GridBagLayout());
15
16         .
17         .
18         .
19
20
21
22
23
24
25
26
27
28     public void actionPerformed(ActionEvent e) {
29         if (e.getSource().equals(submitButton)) {
30             if (!authorField.getText().equals("")) {
31                 Author a = new Author (authorField.getText());
32                 textArea.append(a.toString() + "\n");
33             }
34         }
35     }
36 }
```

# Example



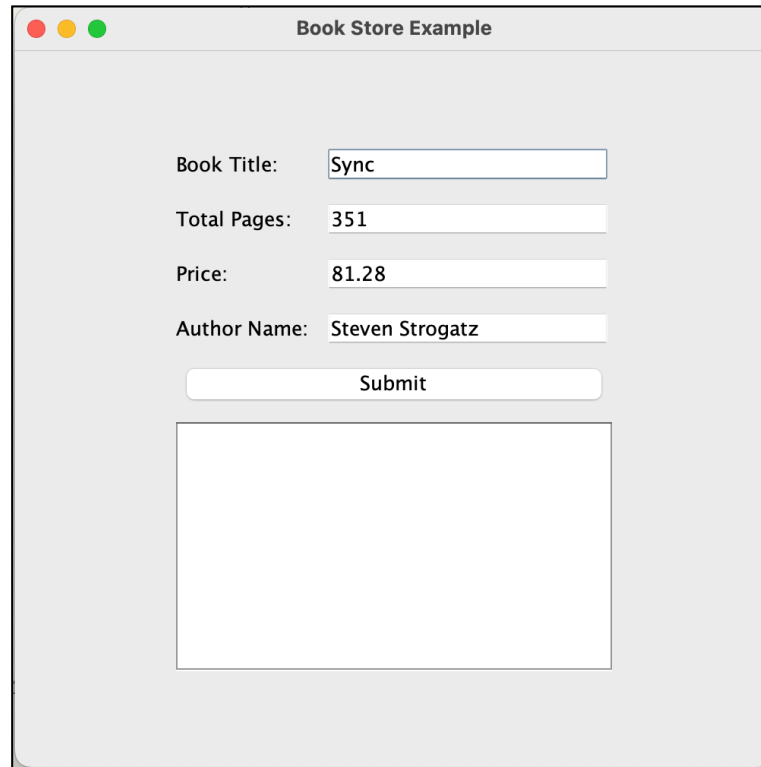
A Java Swing window titled "Book Store Example" with a light gray background. It contains four text input fields labeled "Book Title:", "Total Pages:", "Price:", and "Author Name:". Below these fields is a "Submit" button. At the bottom of the window is a large, empty white text area.



The same "Book Store Example" window after the "Submit" button has been clicked. The "Author Name" field now contains the text "Steven Strogatz". The text area at the bottom now displays the text "AUTHOR: Steven Strogatz" followed by "Books authored:0" on the next line.

```
77
78 public void actionPerformed(ActionEvent e) {
79     if (e.getSource().equals(submitButton)) {
80         if (!authorField.getText().equals("")) {
81             Author a = new Author (authorField.getText());
82
83             textArea.append(a.toString() + "\n");
84         }
85     }
86 }
```

# Example



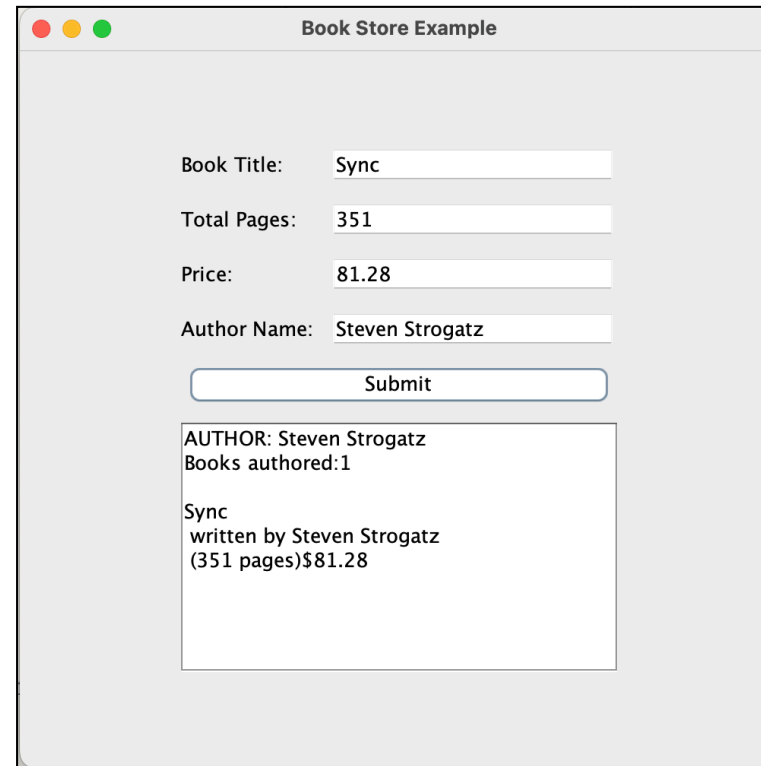
Book Store Example

Book Title:

Total Pages:

Price:

Author Name:



Book Store Example

Book Title:

Total Pages:

Price:

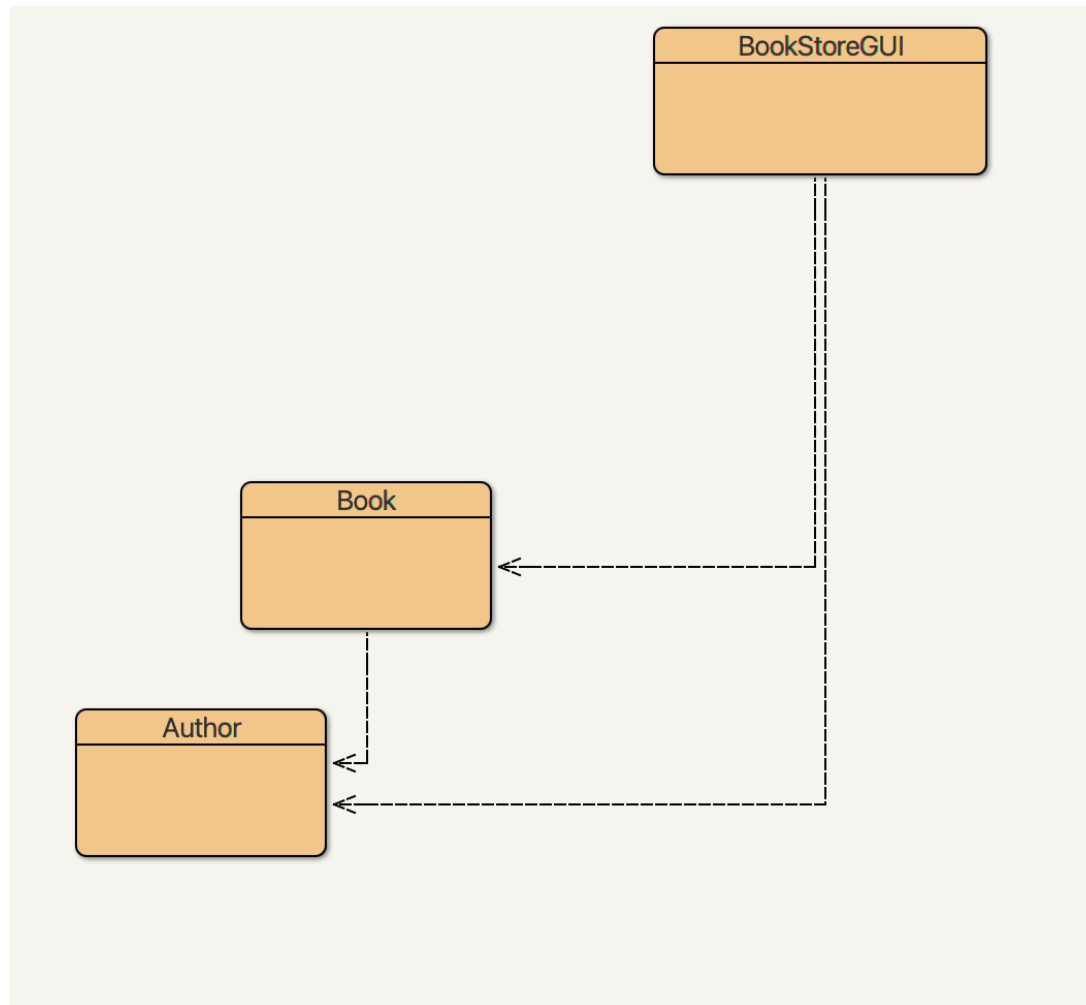
Author Name:

AUTHOR: Steven Strogatz  
Books authored:1

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

```
77 private boolean fieldsNotBlank(){
78     return !authorField.getText().equals("") &&
79            !pageNumField.getText().equals("") &&
80            !priceField.getText().equals("") &&
81            !titleField.getText().equals("");
82 }
83 public void actionPerformed(ActionEvent e) {
84     if (e.getSource().equals(submitButton)) {
85         if(fieldsNotBlank()){
86             Author a = new Author (authorField.getText());
87             Book b = new Book(titleField.getText(), Integer.parseInt(pageNumField.getText()), a);
88             b.setPrice(Double.parseDouble(priceField.getText()));
89             a.addNewBook();
90             textArea.append(a.toString() + "\n" + b.toString());
91         }
92     }
93 }
```

# Example - Tight Coupling





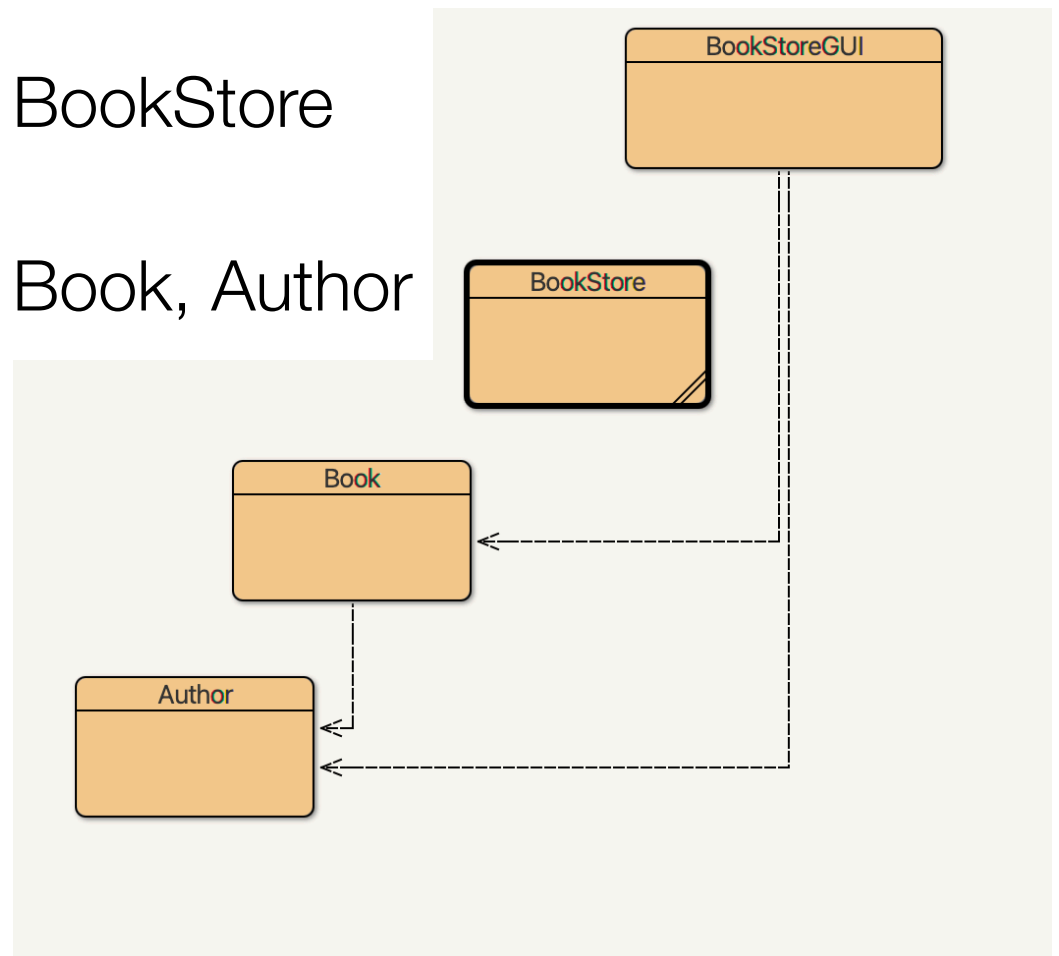
# Applying Model View Separation

User Services

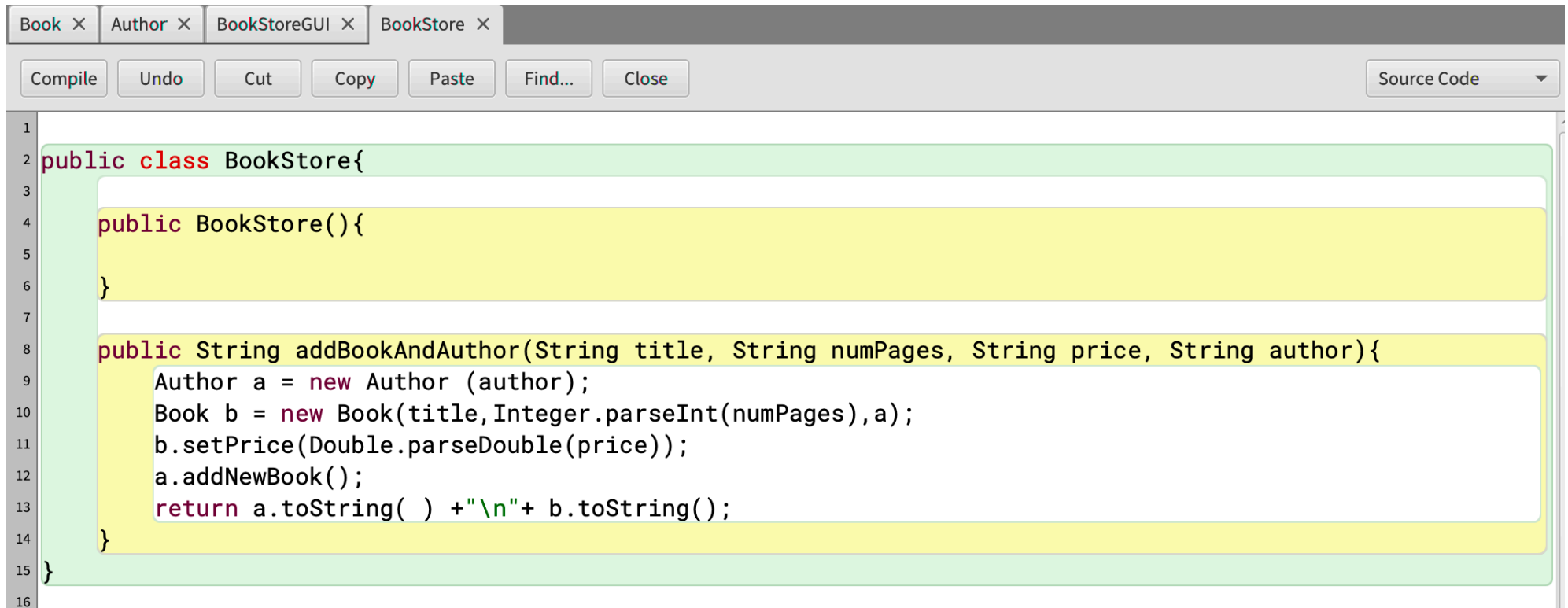
Business Services

Data Services

- BookStoreGUI
- BookStore
- Book, Author



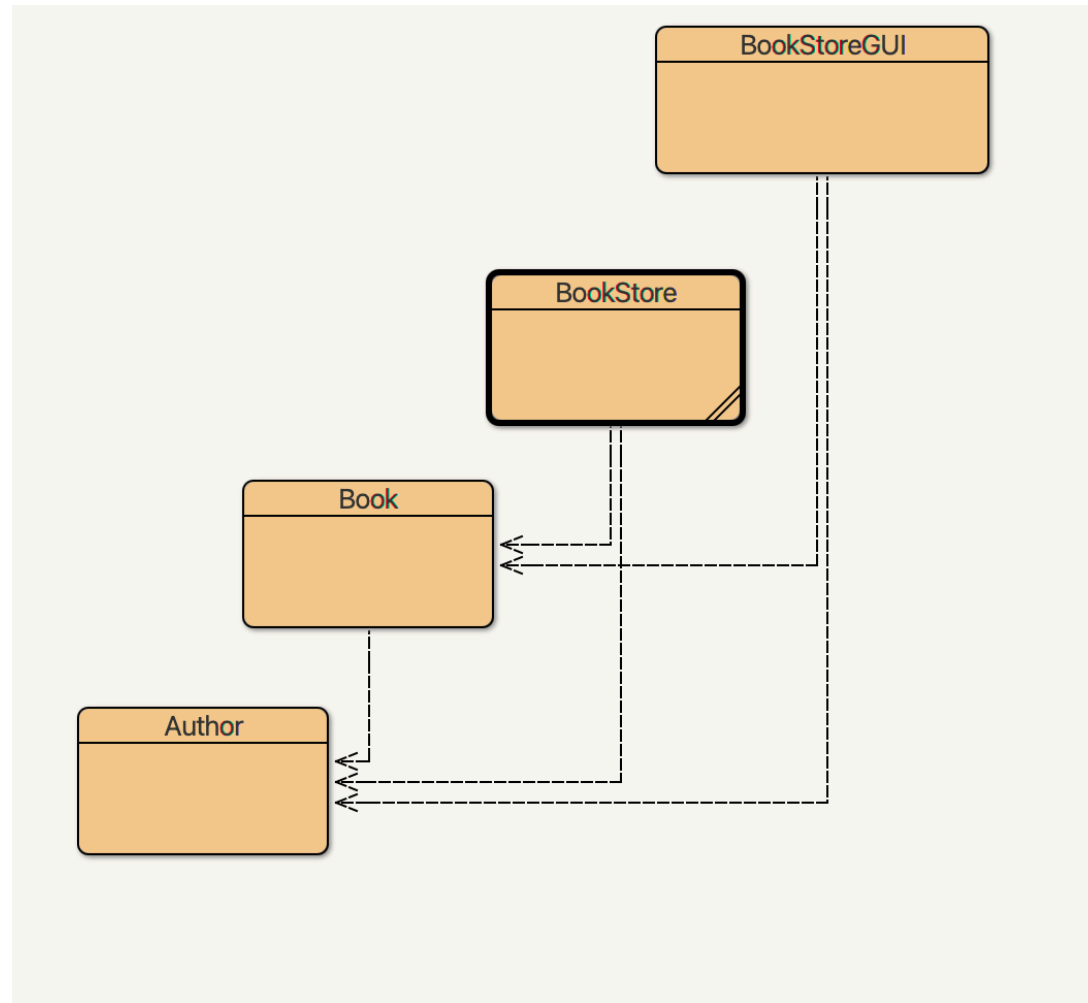
# Applying Model View Separation



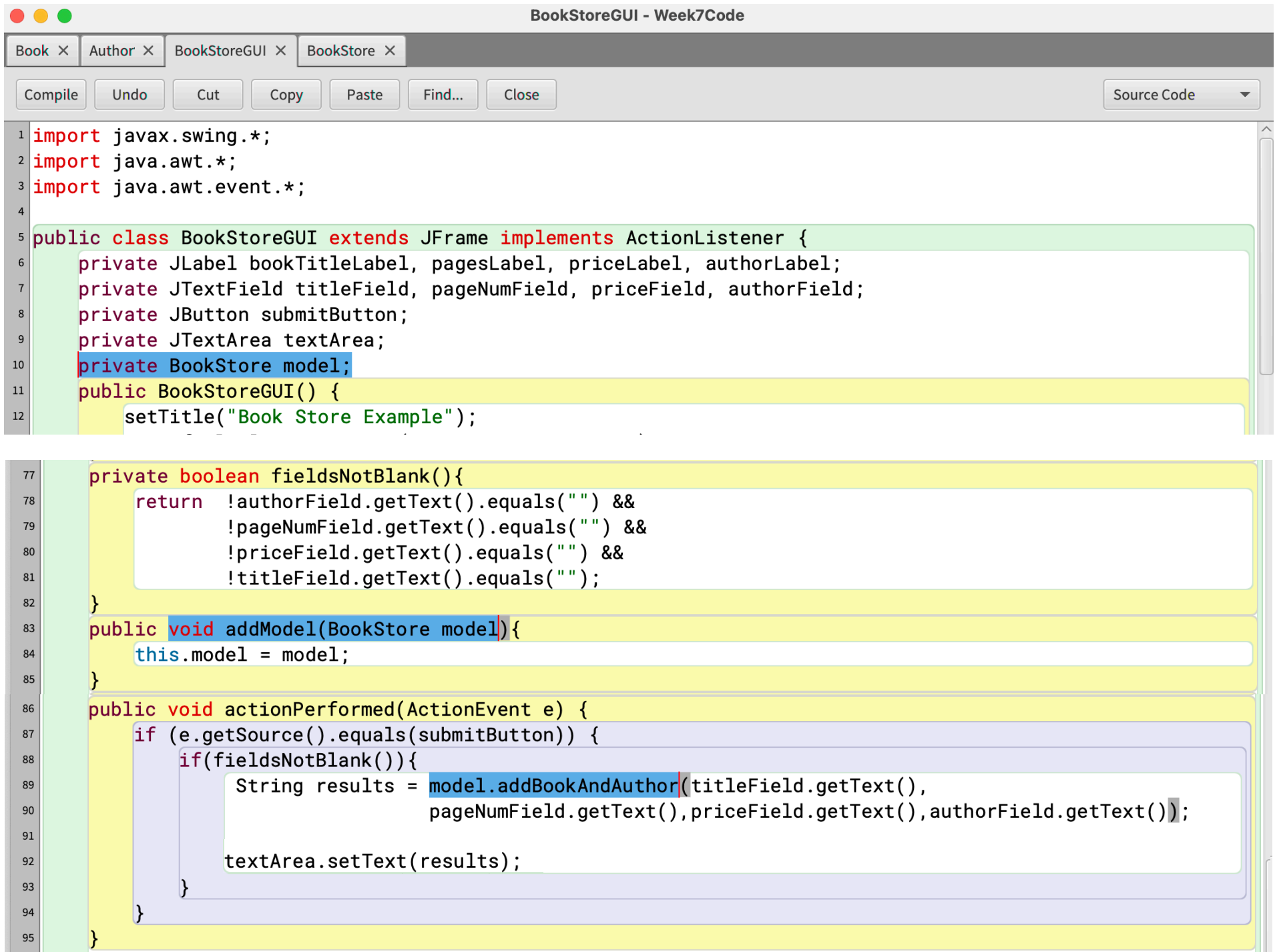
The screenshot shows an IDE window with four tabs: 'Book', 'Author', 'BookStoreGUI', and 'BookStore'. The 'BookStore' tab is active. The toolbar includes buttons for 'Compile', 'Undo', 'Cut', 'Copy', 'Paste', 'Find...', and 'Close'. A 'Source Code' dropdown menu is on the right. The code editor displays the following Java code:

```
1
2 public class BookStore{
3
4     public BookStore(){
5
6     }
7
8     public String addBookAndAuthor(String title, String numPages, String price, String author){
9         Author a = new Author (author);
10        Book b = new Book(title,Integer.parseInt(numPages),a);
11        b.setPrice(Double.parseDouble(price));
12        a.addNewBook();
13        return a.toString( ) +"\n"+ b.toString();
14    }
15 }
16
```

# Decoupling of View to Data Layers

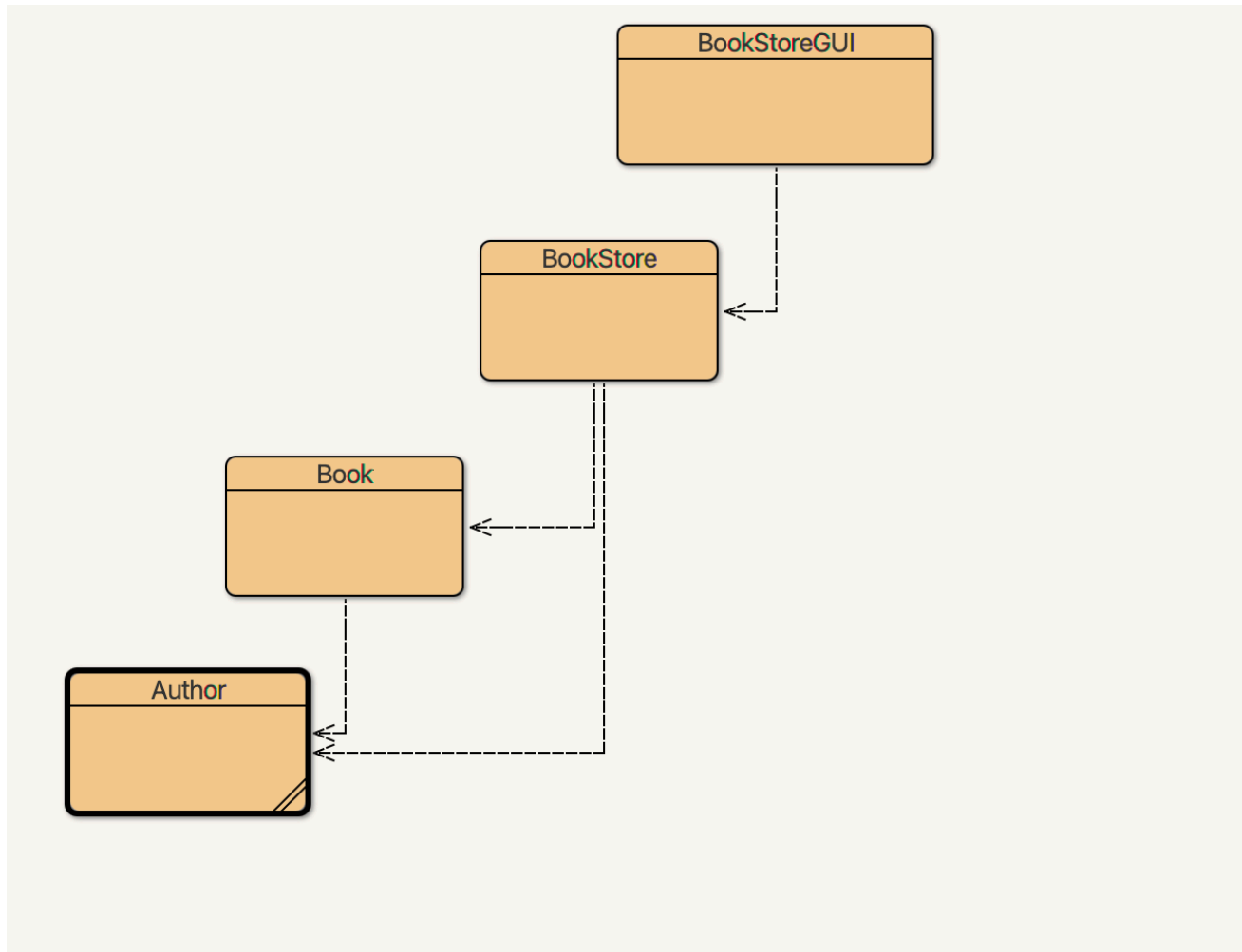


# Applying Model View Separation



```
1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.*;
4
5 public class BookStoreGUI extends JFrame implements ActionListener {
6     private JLabel bookTitleLabel, pagesLabel, priceLabel, authorLabel;
7     private JTextField titleField, pageNumField, priceField, authorField;
8     private JButton submitButton;
9     private JTextArea textArea;
10    private BookStore model;
11    public BookStoreGUI() {
12        setTitle("Book Store Example");
13
14        // ... (code omitted) ...
15
16        private boolean fieldsNotBlank(){
17            return !authorField.getText().equals("") &&
18                   !pageNumField.getText().equals("") &&
19                   !priceField.getText().equals("") &&
20                   !titleField.getText().equals("");
21        }
22
23        public void addModel(BookStore model){
24            this.model = model;
25        }
26
27        public void actionPerformed(ActionEvent e) {
28            if (e.getSource().equals(submitButton)) {
29                if(fieldsNotBlank()){
30                    String results = model.addBookAndAuthor(titleField.getText(),
31                                                            pageNumField.getText(), priceField.getText(), authorField.getText());
32                    textArea.setText(results);
33                }
34            }
35        }
36    }
37 }
```

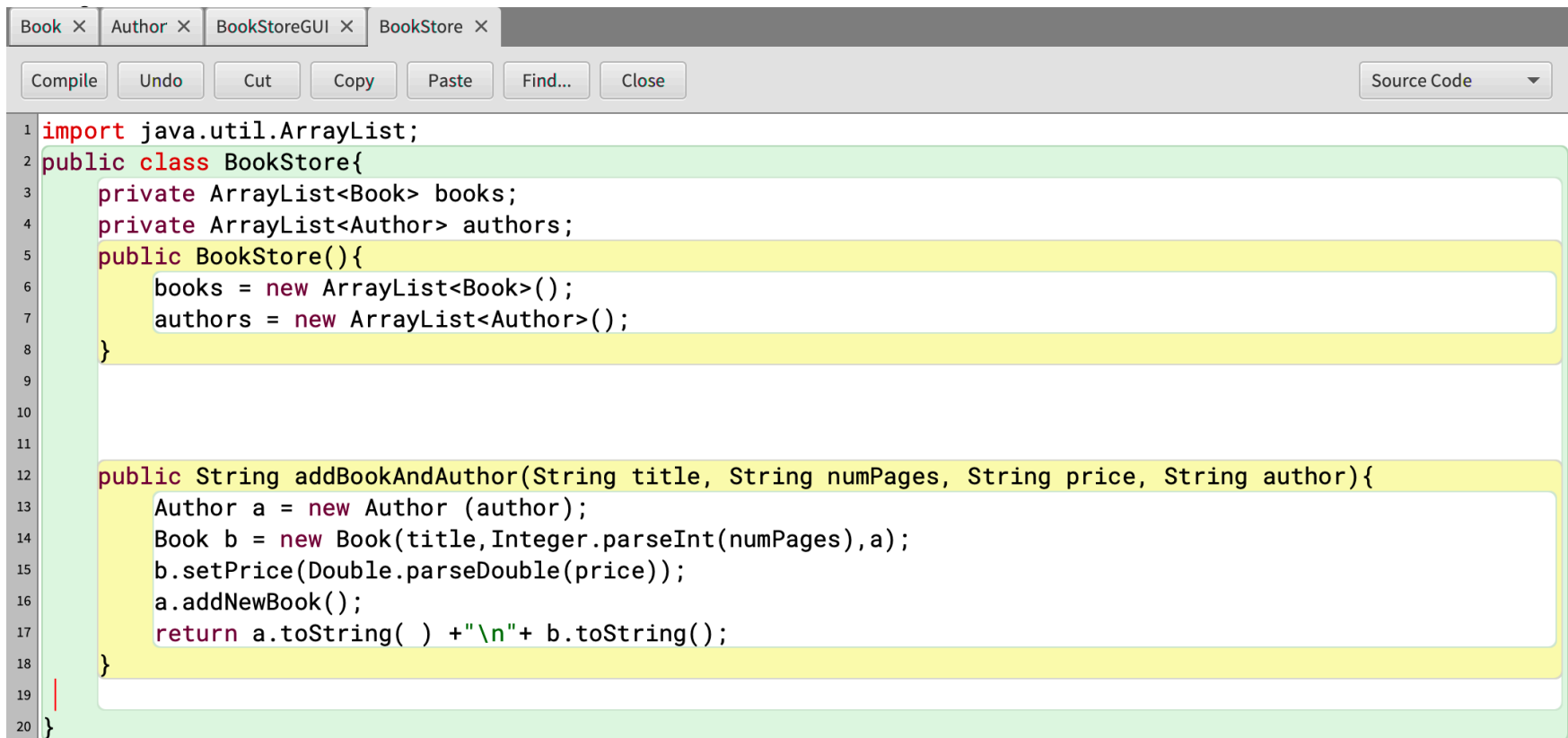
# Applying Model View Separation



# Applying Model View Separation

## Tweaks

- Add equals(..) to Book and Author



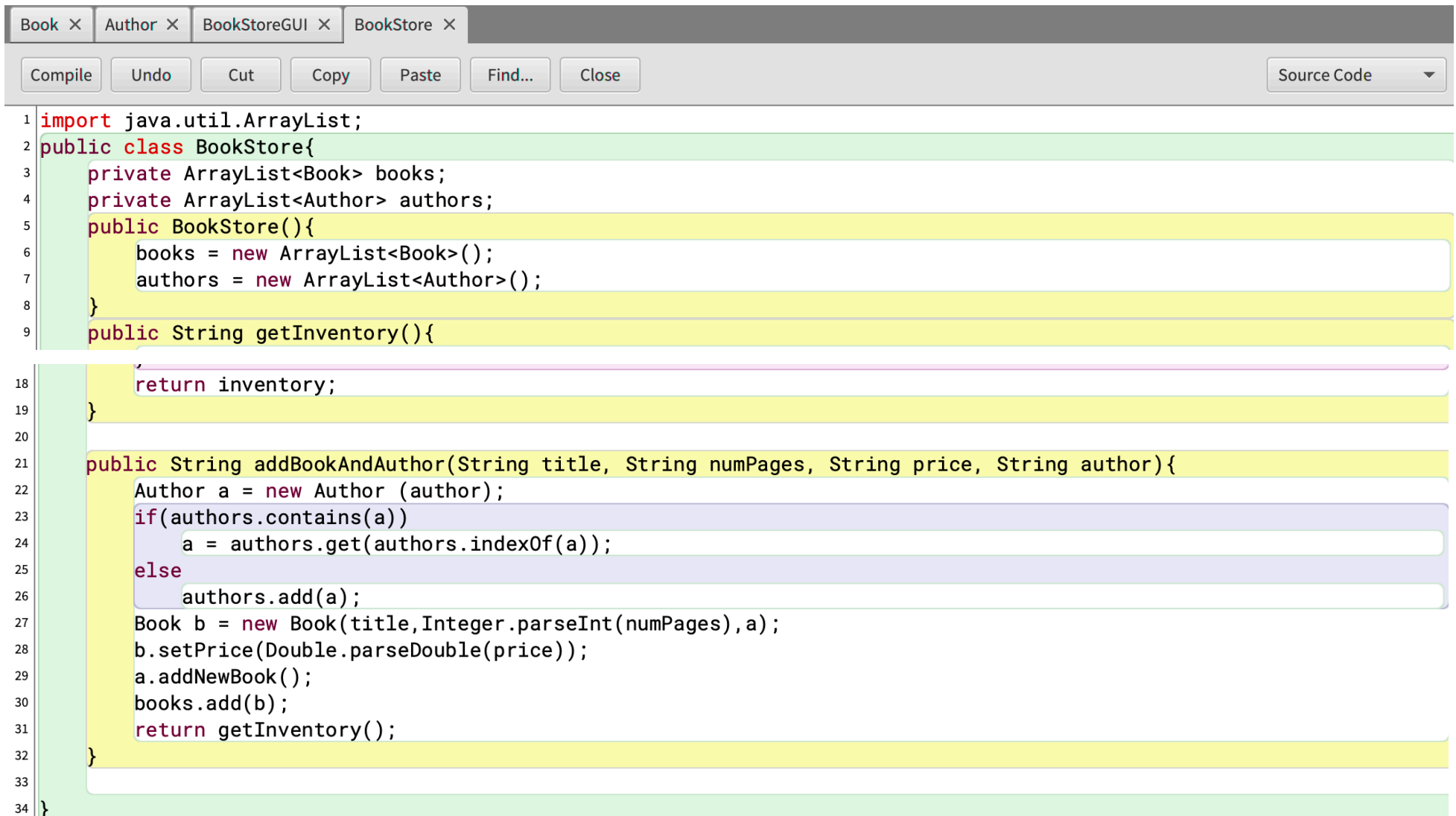
The screenshot shows an IDE window with four tabs: 'Book', 'Author', 'BookStoreGUI', and 'BookStore'. The 'BookStore' tab is active, displaying the source code for the `BookStore` class. The code is as follows:

```
1 import java.util.ArrayList;
2 public class BookStore{
3     private ArrayList<Book> books;
4     private ArrayList<Author> authors;
5     public BookStore(){
6         books = new ArrayList<Book>();
7         authors = new ArrayList<Author>();
8     }
9
10
11
12     public String addBookAndAuthor(String title, String numPages, String price, String author){
13         Author a = new Author (author);
14         Book b = new Book(title,Integer.parseInt(numPages),a);
15         b.setPrice(Double.parseDouble(price));
16         a.addNewBook();
17         return a.toString( ) + "\n" + b.toString();
18     }
19
20 }
```

# Applying Model View Separation

## Tweaks

- Add equals(..) to Book and Author



```
1 import java.util.ArrayList;
2 public class BookStore{
3     private ArrayList<Book> books;
4     private ArrayList<Author> authors;
5     public BookStore(){
6         books = new ArrayList<Book>();
7         authors = new ArrayList<Author>();
8     }
9     public String getInventory(){
10
11     }
12     return inventory;
13 }
14
15 public String addBookAndAuthor(String title, String numPages, String price, String author){
16     Author a = new Author (author);
17     if(authors.contains(a))
18         a = authors.get(authors.indexOf(a));
19     else
20         authors.add(a);
21     Book b = new Book(title,Integer.parseInt(numPages),a);
22     b.setPrice(Double.parseDouble(price));
23     a.addNewBook();
24     books.add(b);
25     return getInventory();
26 }
27
28 }
```

# Applying Model View Separation

```
5 public class BookStoreGUI extends JFrame implements ActionListener {  
6     private JLabel bookTitleLabel, pagesLabel, priceLabel, authorLabel;  
7     private JTextField titleField, pageNumField, priceField, authorField;  
8     private JButton submitButton;  
9     private JTextArea textArea;  
10    private BookStore model;  
11  
12    public BookStoreGUI() {
```

```
96  
97    public static void main(String[] args) {  
98        BookStoreGUI gui = new BookStoreGUI();  
99        BookStore model = new BookStore();  
100        gui.addModel(model);  
101        gui.setSize(500,500);  
102    }  
103 }  
104
```



# Applying Model View Separation

The image shows three overlapping windows of a 'Book Store Example' application, illustrating the progression of data entry and display. Each window has a title bar with three colored buttons (red, yellow, green) and a title 'Book Store Example'.

**Leftmost Window:**

- Book Title: Sync
- Total Pages: 351
- Price: 81.28
- Author Name: Steven Strogatz
- Submit button
- Display area:  
AUTHOR: Steven Strogatz  
Books authored:1  
Sync  
written by Steven Strogatz  
(351 pages)\$81.28

**Middle Window:**

- Book Title: The Joy of X
- Total Pages: 336
- Price: 94.82
- Author Name: Steven Strogatz
- Submit button
- Display area:  
AUTHOR: Steven Strogatz  
Books authored:1  
Sync  
written by Steven Strogatz  
(351 pages)\$81.28

**Rightmost Window:**

- Book Title: The Joy of X
- Total Pages: 336
- Price: 94.82
- Author Name: Steven Strogatz
- Submit button
- Display area:  
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)

