

# Relationships between Objects

Dependencies, Associations and Generalisations

COMP2603  
Object Oriented Programming 1

Week 3, Lecture 2



# Outline

- Variable vs Object Equality
- Types of Relationships in Object-Oriented Programming
  - Dependencies
  - Associations
  - Generalisations

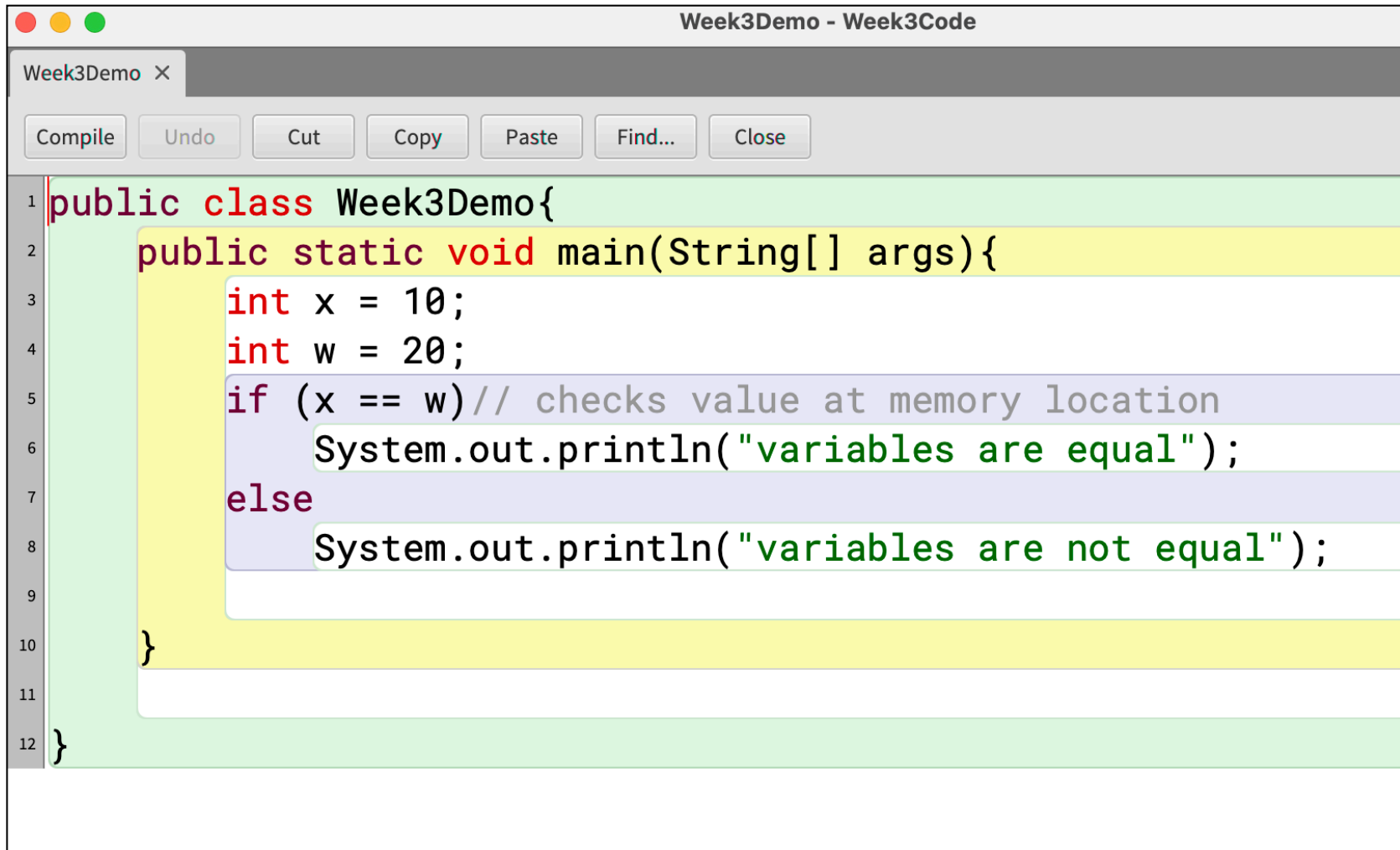
# Variable Assignment vs Equality

The symbol `=` is the assignment operator. It assigns the value on its right-hand side to the variable on its left-hand side.

The symbol `==` is the equality operator. It evaluates whether the expressions on its left- and right-hand sides have the same value and returns either `true` or `false`.

<https://runestone.academy/ns/books/published/javajavajava/java-language-elements.html>

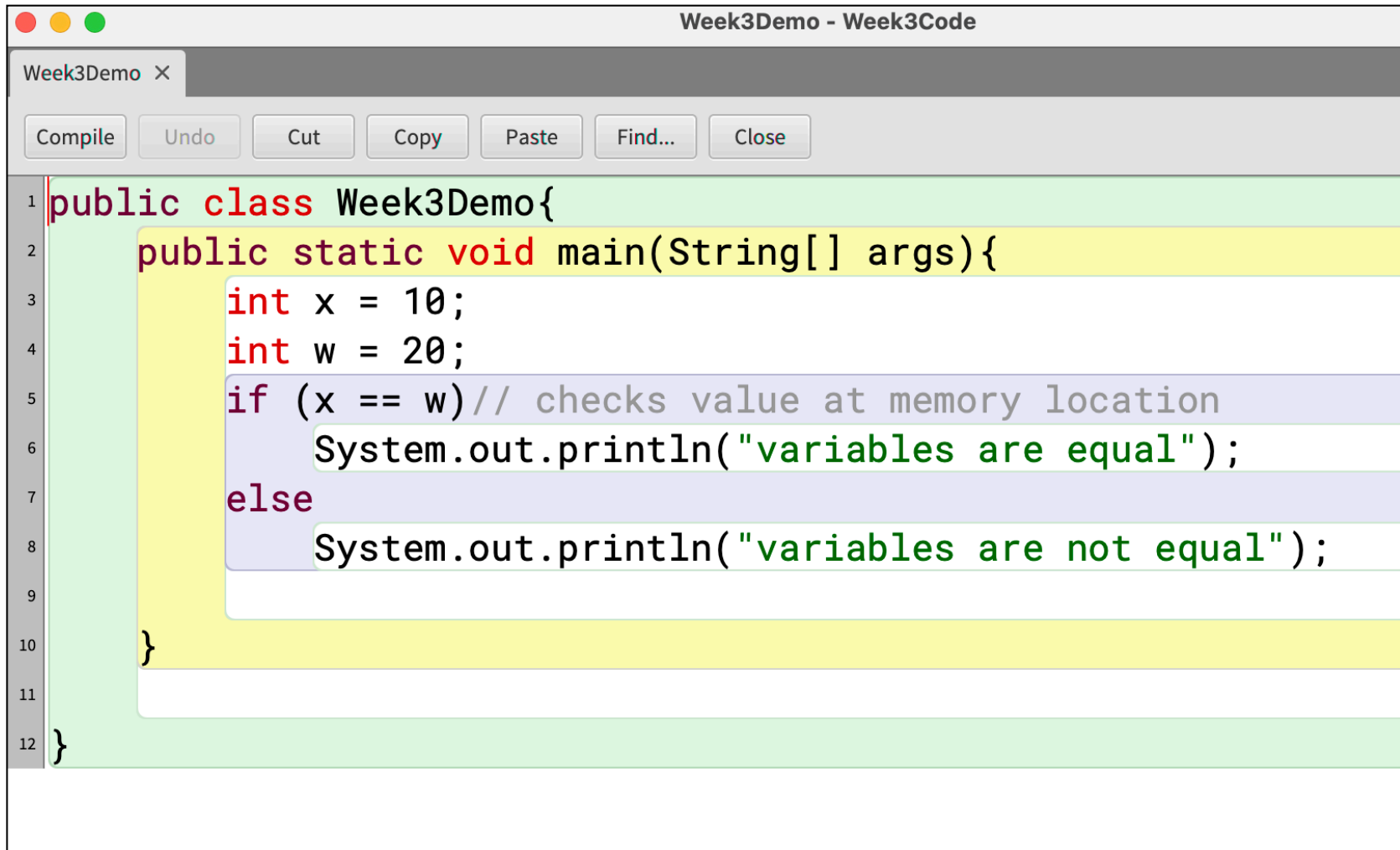
# Variable Equality - Example 1



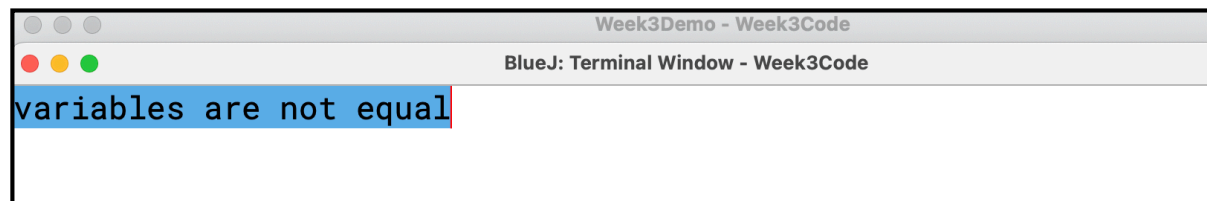
```
1 public class Week3Demo{
2     public static void main(String[] args){
3         int x = 10;
4         int w = 20;
5         if (x == w) // checks value at memory location
6             System.out.println("variables are equal");
7         else
8             System.out.println("variables are not equal");
9     }
10 }
11
12 }
```

What is printed?  
(A) variables are equal  
(B) variables are not equal

# Variable Equality - Example 1

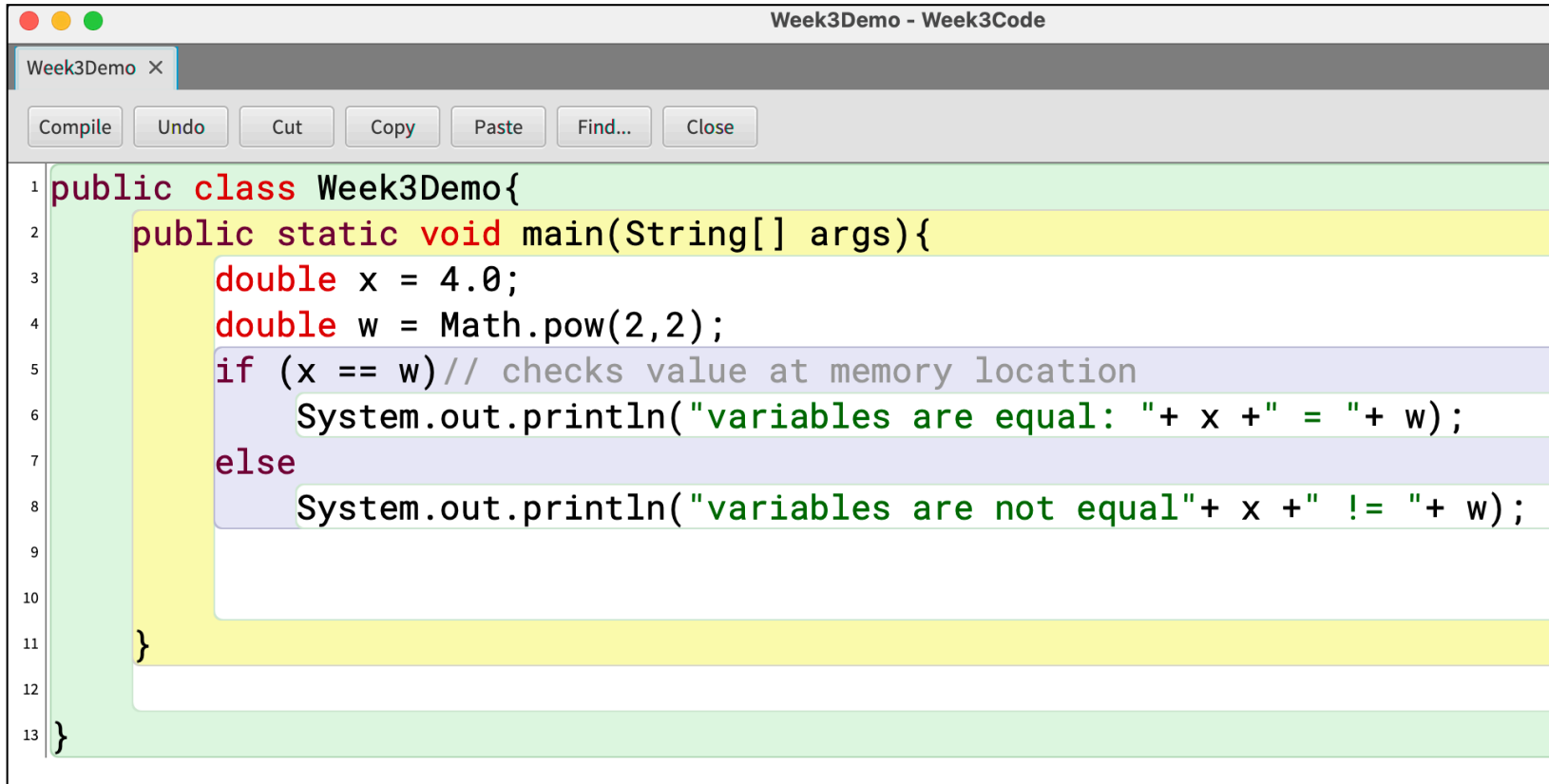


```
1 public class Week3Demo{
2     public static void main(String[] args){
3         int x = 10;
4         int w = 20;
5         if (x == w) // checks value at memory location
6             System.out.println("variables are equal");
7         else
8             System.out.println("variables are not equal");
9     }
10 }
11
12 }
```



```
BlueJ: Terminal Window - Week3Code
variables are not equal
```

## Variable Equality - Example 2

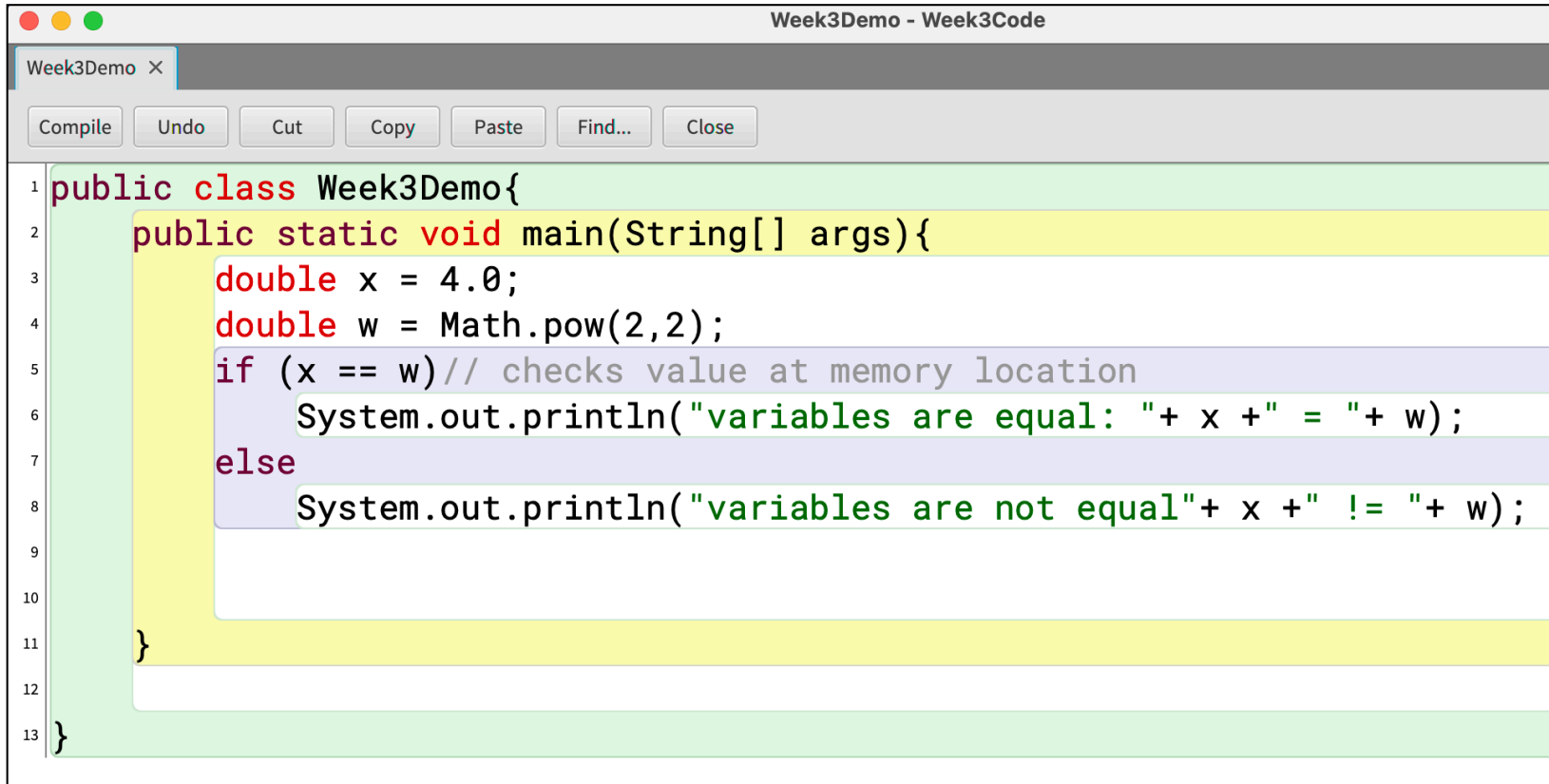


```
1 public class Week3Demo{
2     public static void main(String[] args){
3         double x = 4.0;
4         double w = Math.pow(2,2);
5         if (x == w) // checks value at memory location
6             System.out.println("variables are equal: " + x + " = " + w);
7         else
8             System.out.println("variables are not equal" + x + " != " + w);
9
10
11     }
12
13 }
```

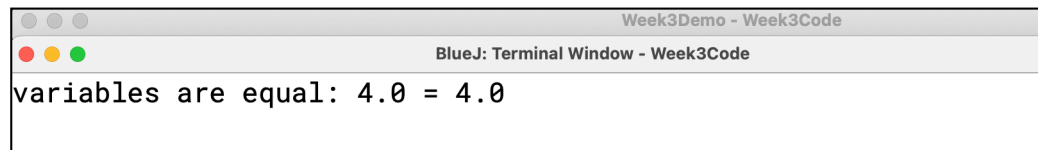
What is printed?

- (A) variables are not equal 4.0 != 2
- (B) variables are not equal 4.0 != 4
- (C) variables are equal 2 = 2
- (D) variables are equal 4.0 = 4.0

# Variable Equality - Example 2



```
1 public class Week3Demo{
2     public static void main(String[] args){
3         double x = 4.0;
4         double w = Math.pow(2,2);
5         if (x == w) // checks value at memory location
6             System.out.println("variables are equal: " + x + " = " + w);
7         else
8             System.out.println("variables are not equal" + x + " != " + w);
9
10    }
11 }
12
13 }
```



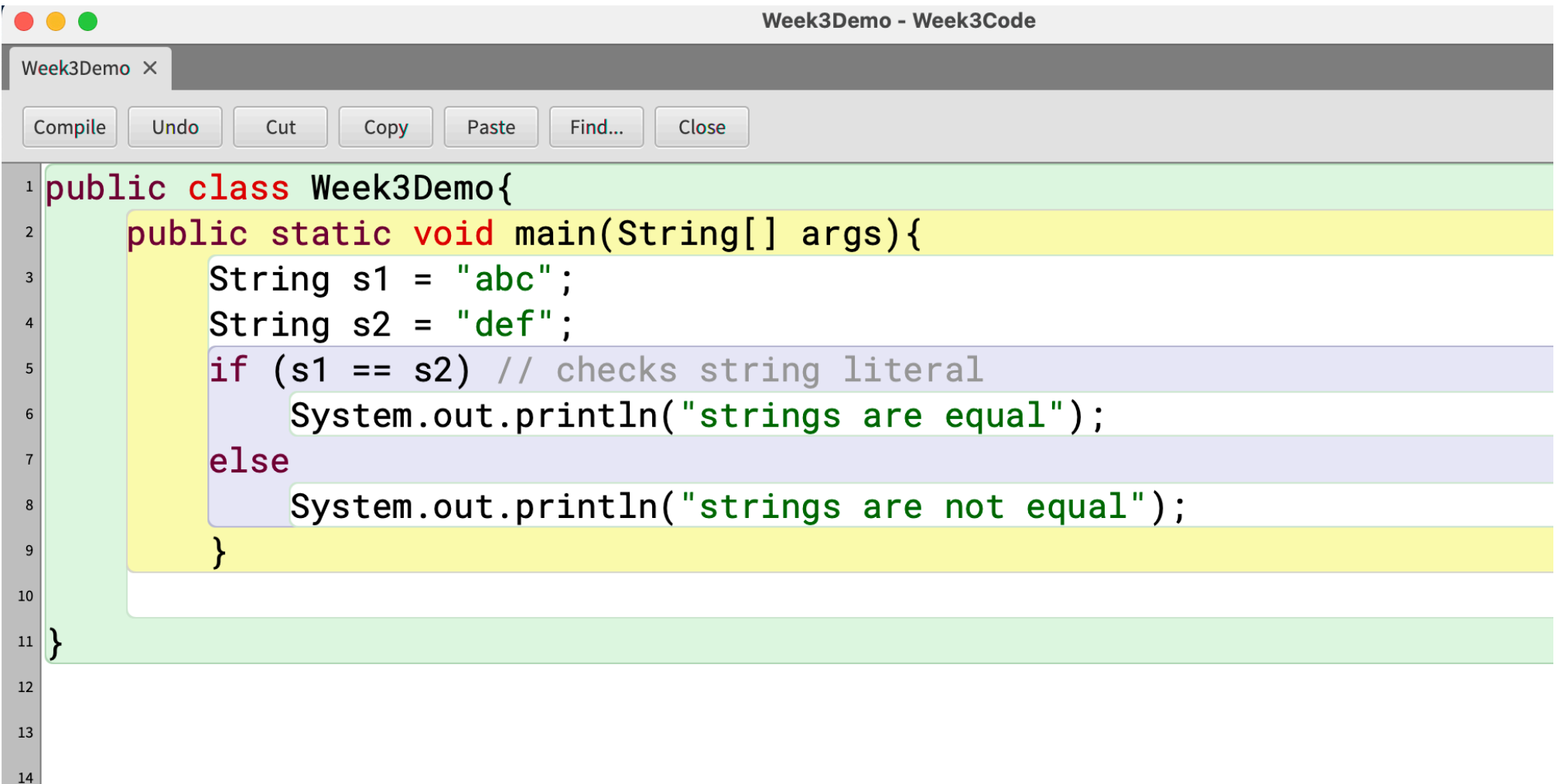
```
Week3Demo - Week3Code
BlueJ: Terminal Window - Week3Code
variables are equal: 4.0 = 4.0
```

# String Equality

The “==” operator only checks the referential equality of two Strings, meaning if they reference the same object or not.



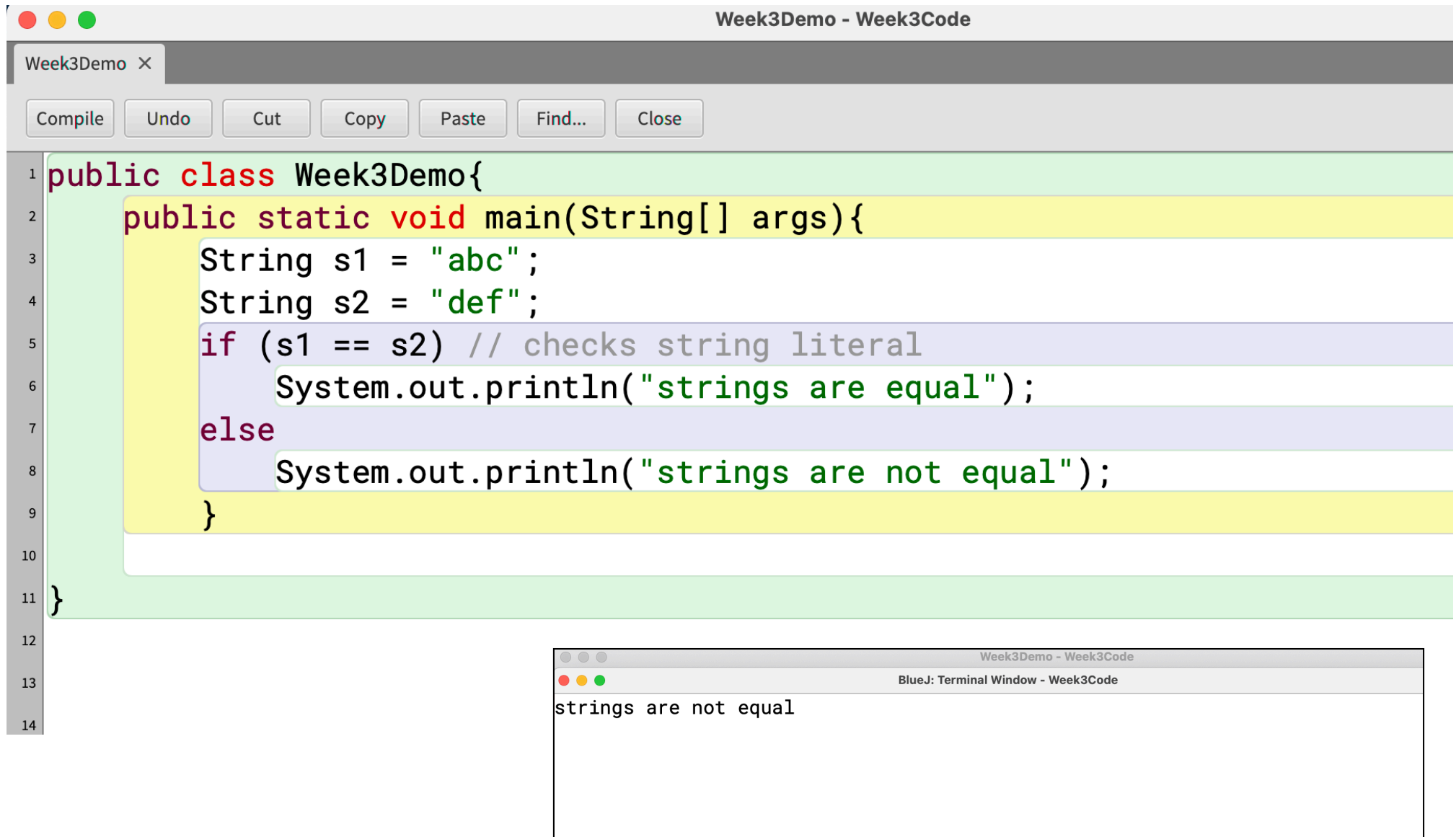
# String Equality - Example 1



The screenshot shows a code editor window with a title bar 'Week3Demo - Week3Code'. Below the title bar is a tab labeled 'Week3Demo X'. A toolbar contains buttons for 'Compile', 'Undo', 'Cut', 'Copy', 'Paste', 'Find...', and 'Close'. The code is as follows:

```
1 public class Week3Demo{
2     public static void main(String[] args){
3         String s1 = "abc";
4         String s2 = "def";
5         if (s1 == s2) // checks string literal
6             System.out.println("strings are equal");
7         else
8             System.out.println("strings are not equal");
9     }
10
11 }
12
13
14
```

# String Equality - Example 1



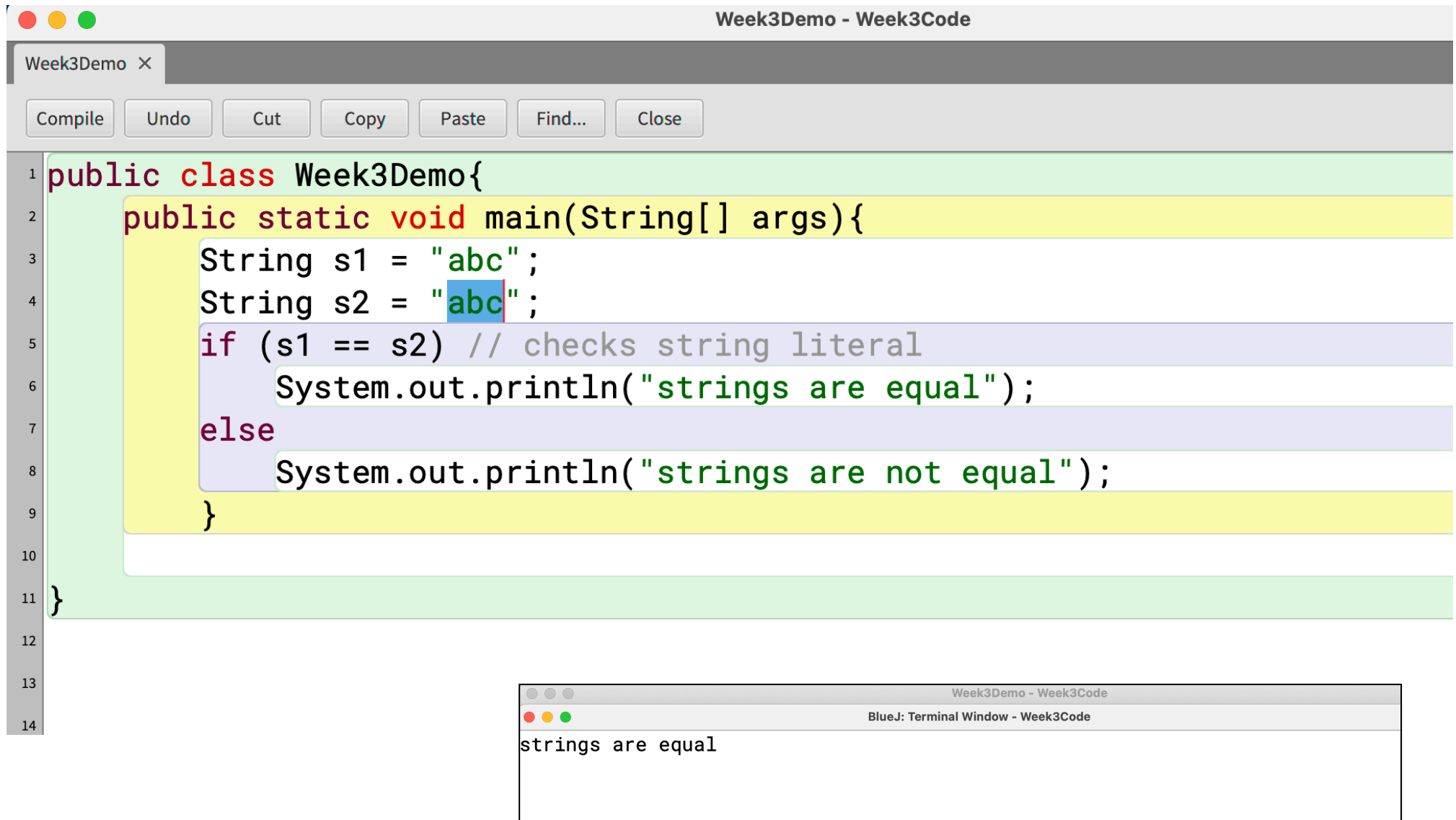
The image shows a Java IDE window titled "Week3Demo - Week3Code". The code editor contains the following Java code:

```
1 public class Week3Demo{
2     public static void main(String[] args){
3         String s1 = "abc";
4         String s2 = "def";
5         if (s1 == s2) // checks string literal
6             System.out.println("strings are equal");
7         else
8             System.out.println("strings are not equal");
9     }
10 }
11
12
13
14
```

Below the code editor, there is a terminal window titled "BlueJ: Terminal Window - Week3Code". The terminal displays the output of the program:

```
strings are not equal
```

## String Equality - Example 2



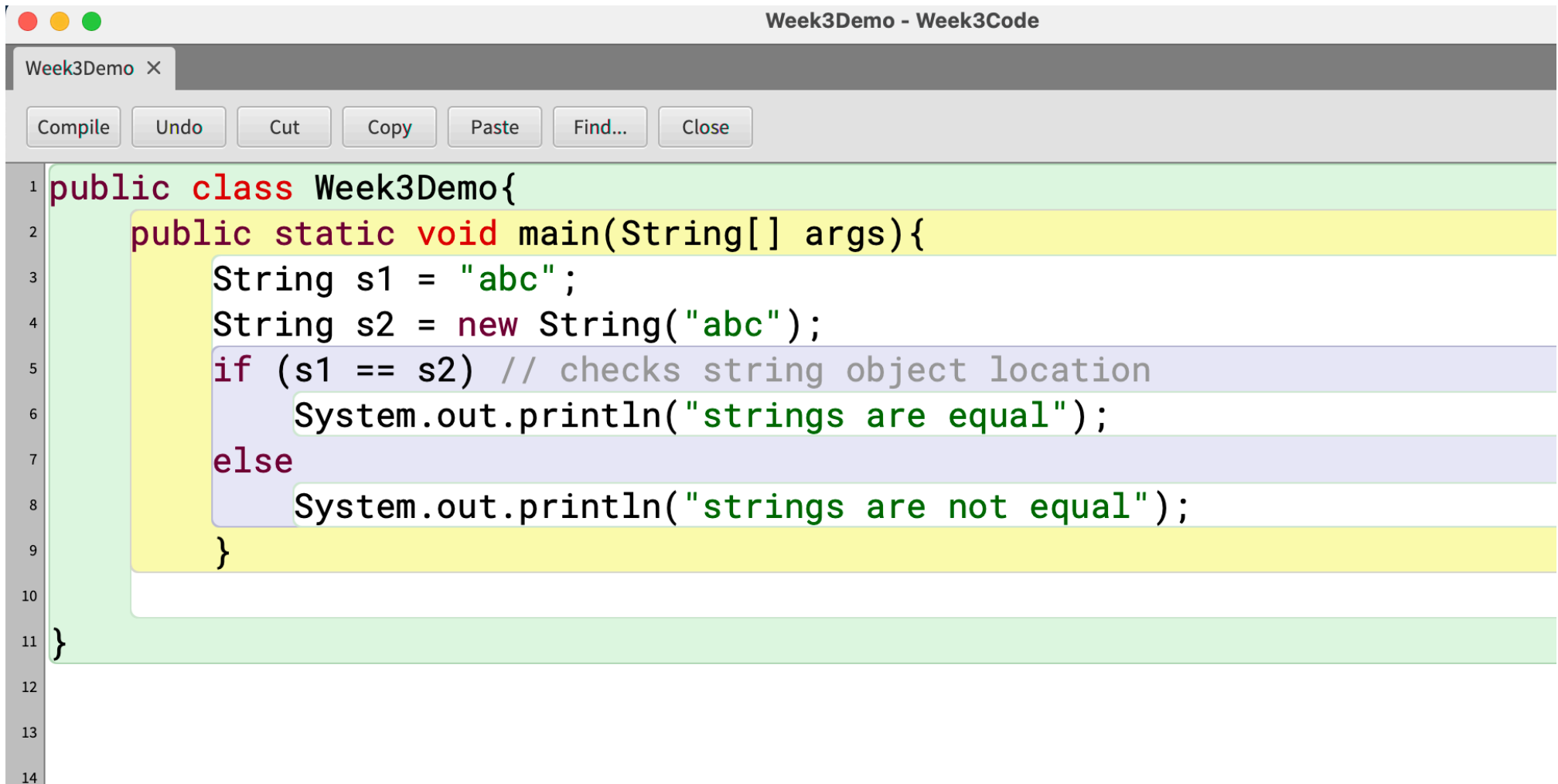
The image shows a Java IDE window titled "Week3Demo - Week3Code". The code editor contains the following Java code:

```
1 public class Week3Demo{
2     public static void main(String[] args){
3         String s1 = "abc";
4         String s2 = "abc";
5         if (s1 == s2) // checks string literal
6             System.out.println("strings are equal");
7         else
8             System.out.println("strings are not equal");
9     }
10 }
11
12
13
14
```

Below the code editor, there is a terminal window titled "BlueJ: Terminal Window - Week3Code". The terminal displays the output of the program:

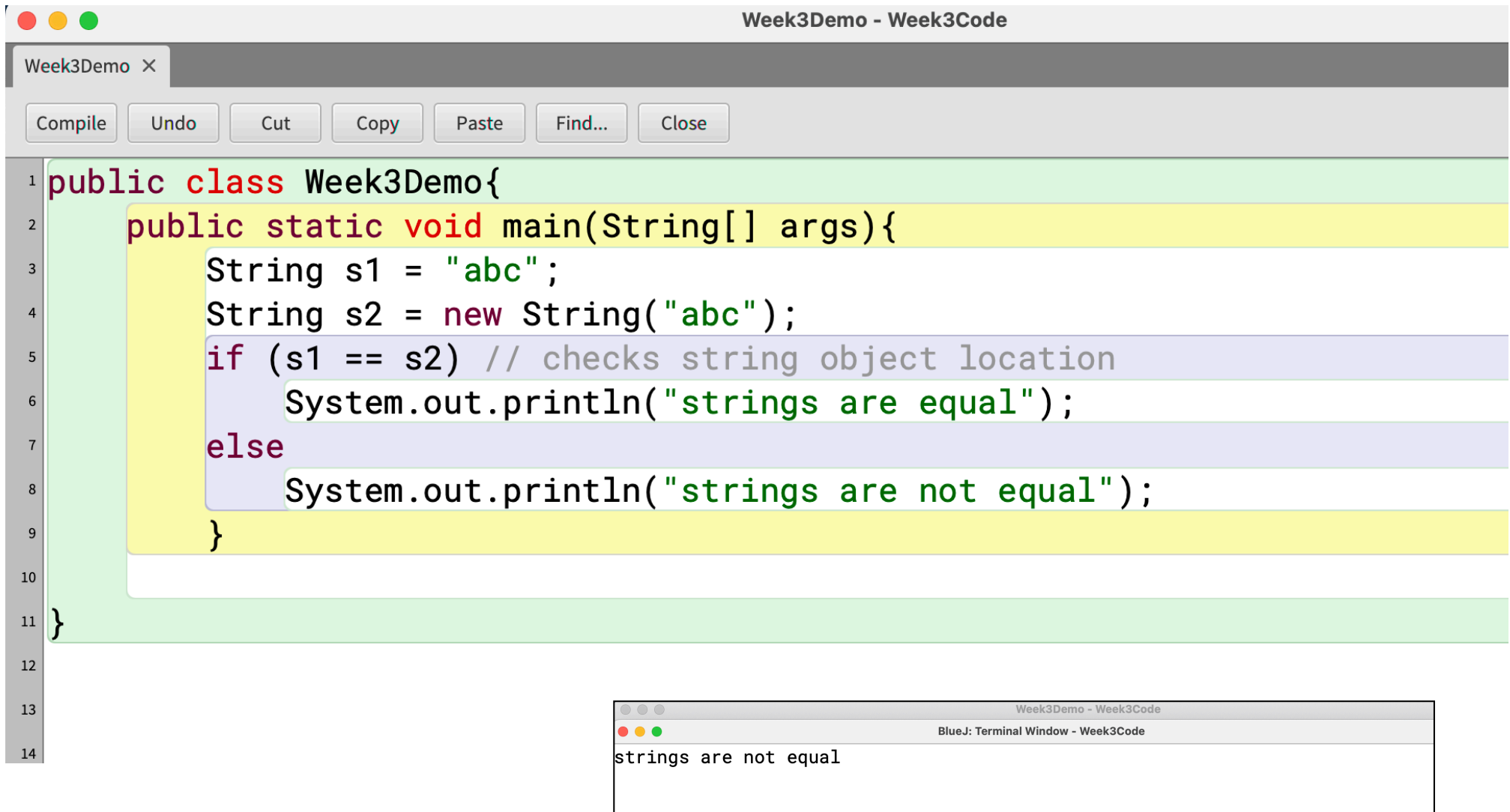
```
strings are equal
```

# String Equality - Example 3



```
1 public class Week3Demo{
2     public static void main(String[] args){
3         String s1 = "abc";
4         String s2 = new String("abc");
5         if (s1 == s2) // checks string object location
6             System.out.println("strings are equal");
7         else
8             System.out.println("strings are not equal");
9     }
10
11 }
12
13
14
```

# String Equality - Example 3



The image shows a code editor window titled "Week3Demo - Week3Code" with a tab labeled "Week3Demo". The editor contains the following Java code:

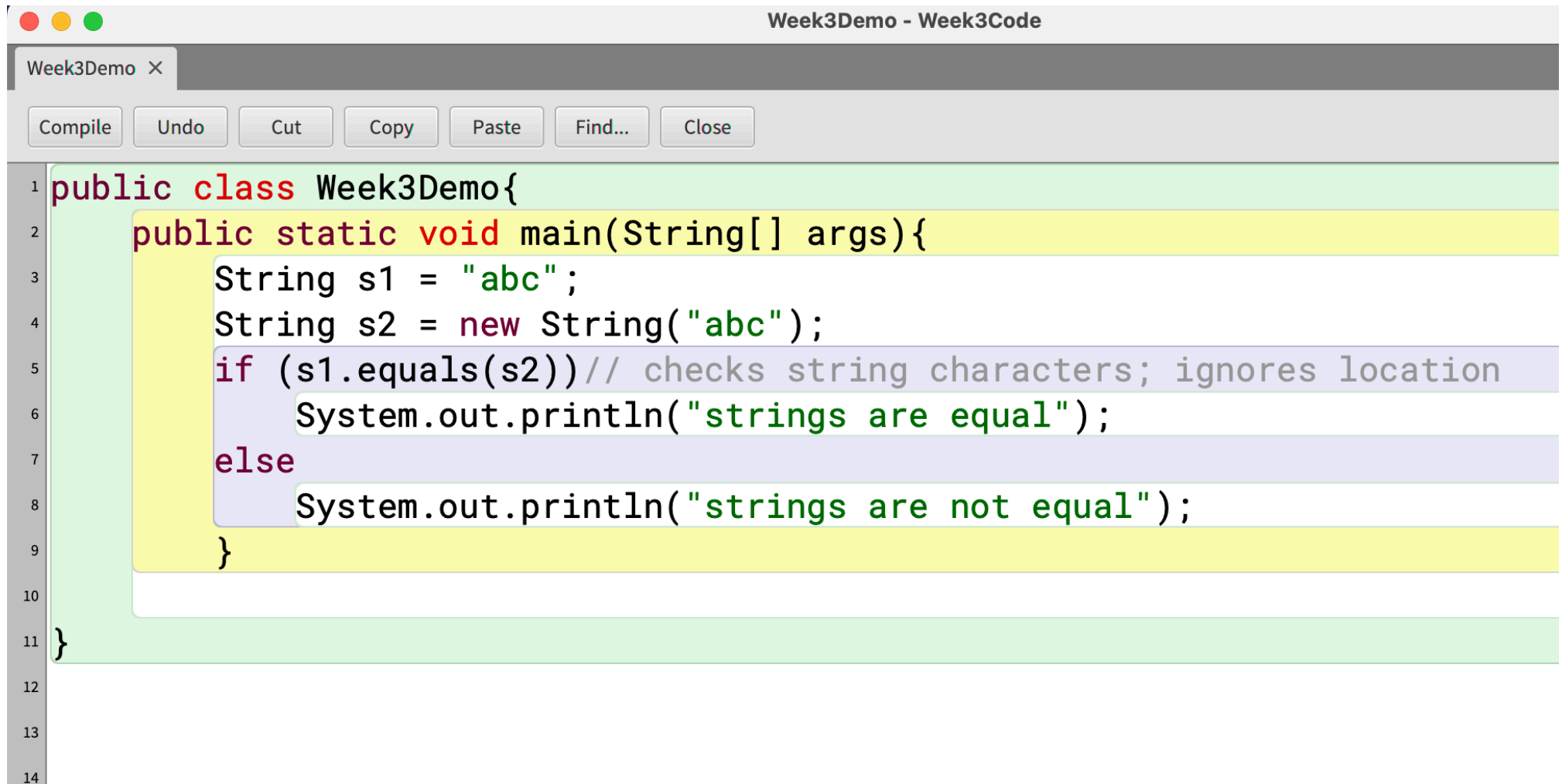
```
1 public class Week3Demo{
2     public static void main(String[] args){
3         String s1 = "abc";
4         String s2 = new String("abc");
5         if (s1 == s2) // checks string object location
6             System.out.println("strings are equal");
7         else
8             System.out.println("strings are not equal");
9     }
10 }
11
12
13
14
```

Below the code editor is a terminal window titled "Week3Demo - Week3Code" with a subtitle "BlueJ: Terminal Window - Week3Code". The terminal displays the output:

```
strings are not equal
```

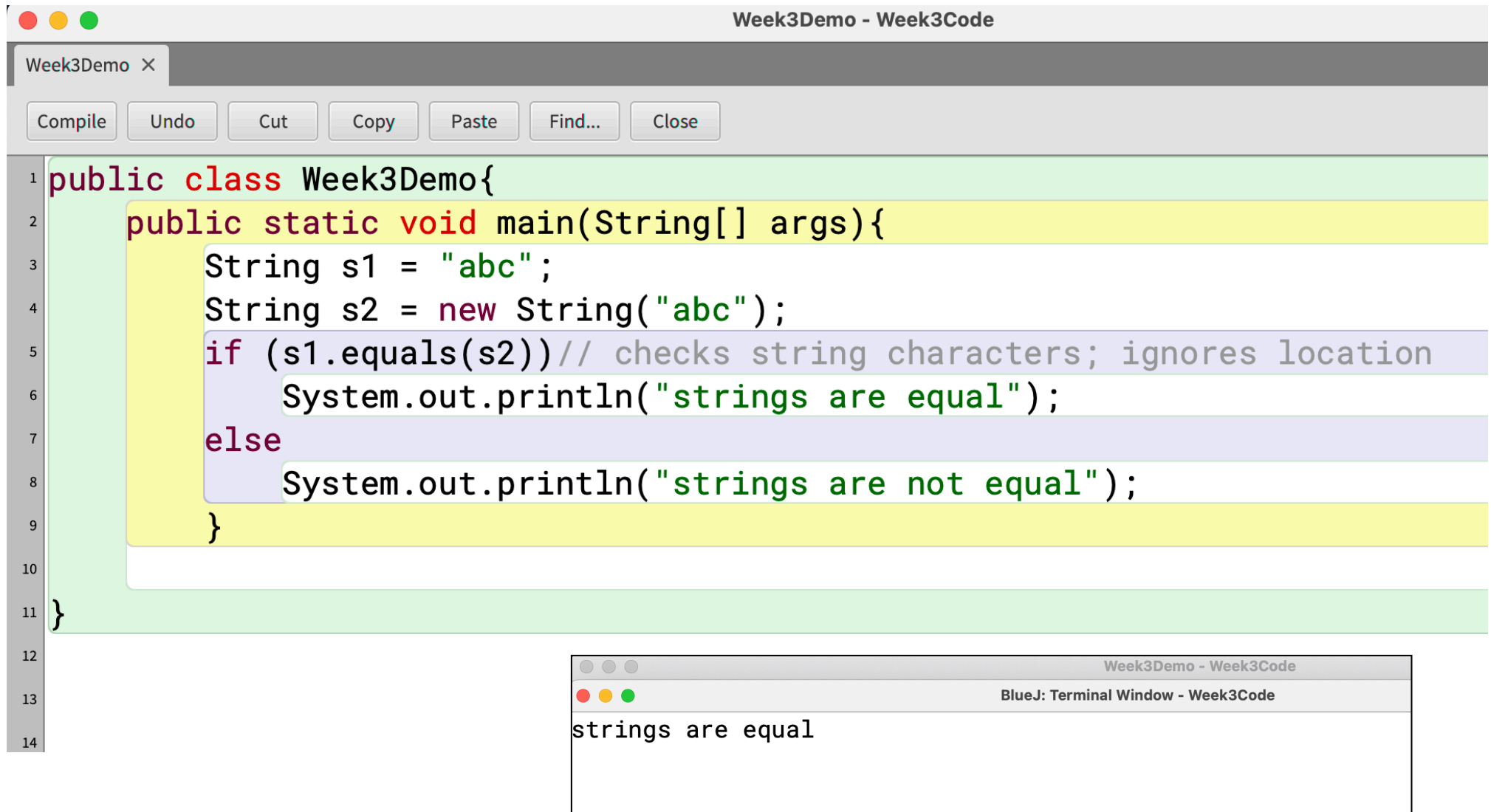
✗ But this output could be conceptually incorrect

# String Equality - Example 4



```
1 public class Week3Demo{
2     public static void main(String[] args){
3         String s1 = "abc";
4         String s2 = new String("abc");
5         if (s1.equals(s2))// checks string characters; ignores location
6             System.out.println("strings are equal");
7         else
8             System.out.println("strings are not equal");
9     }
10
11 }
12
13
14
```

# String Equality - Example 4



The image shows a Java IDE window titled "Week3Demo - Week3Code". The code is as follows:

```
1 public class Week3Demo{
2     public static void main(String[] args){
3         String s1 = "abc";
4         String s2 = new String("abc");
5         if (s1.equals(s2))// checks string characters; ignores location
6             System.out.println("strings are equal");
7         else
8             System.out.println("strings are not equal");
9     }
10 }
11
12
13
14
```

Below the code editor, there is a terminal window titled "BlueJ: Terminal Window - Week3Code" showing the output:

```
strings are equal
```

**Why? Because the String class provides its own equals method**

# Object Equality

Consider the equals( ) method of the Object class:

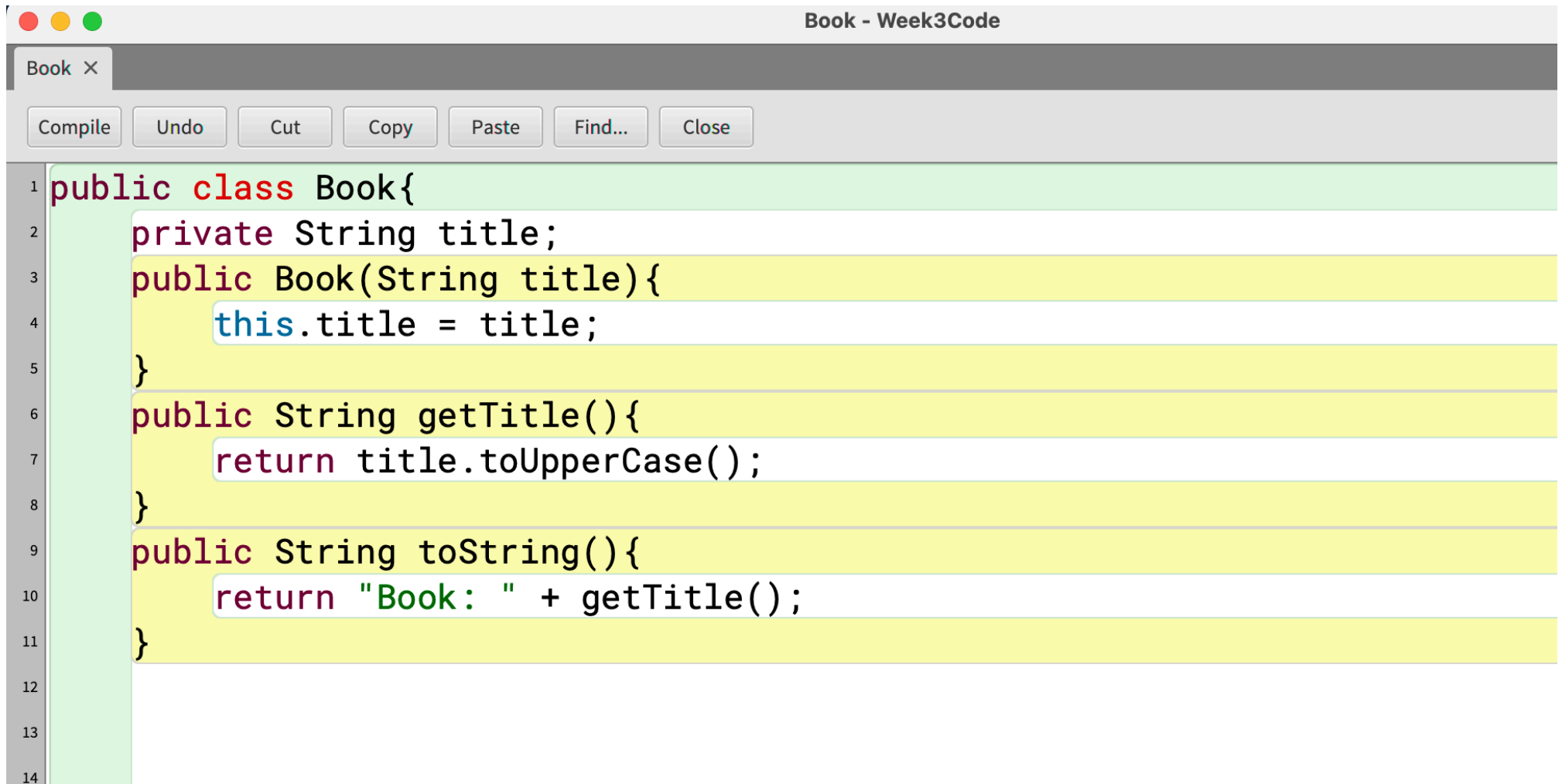
```
public boolean equals(Object obj)
```

This method returns true if the current object is stored at the same memory address as obj and false otherwise.

This method behaves just like ==

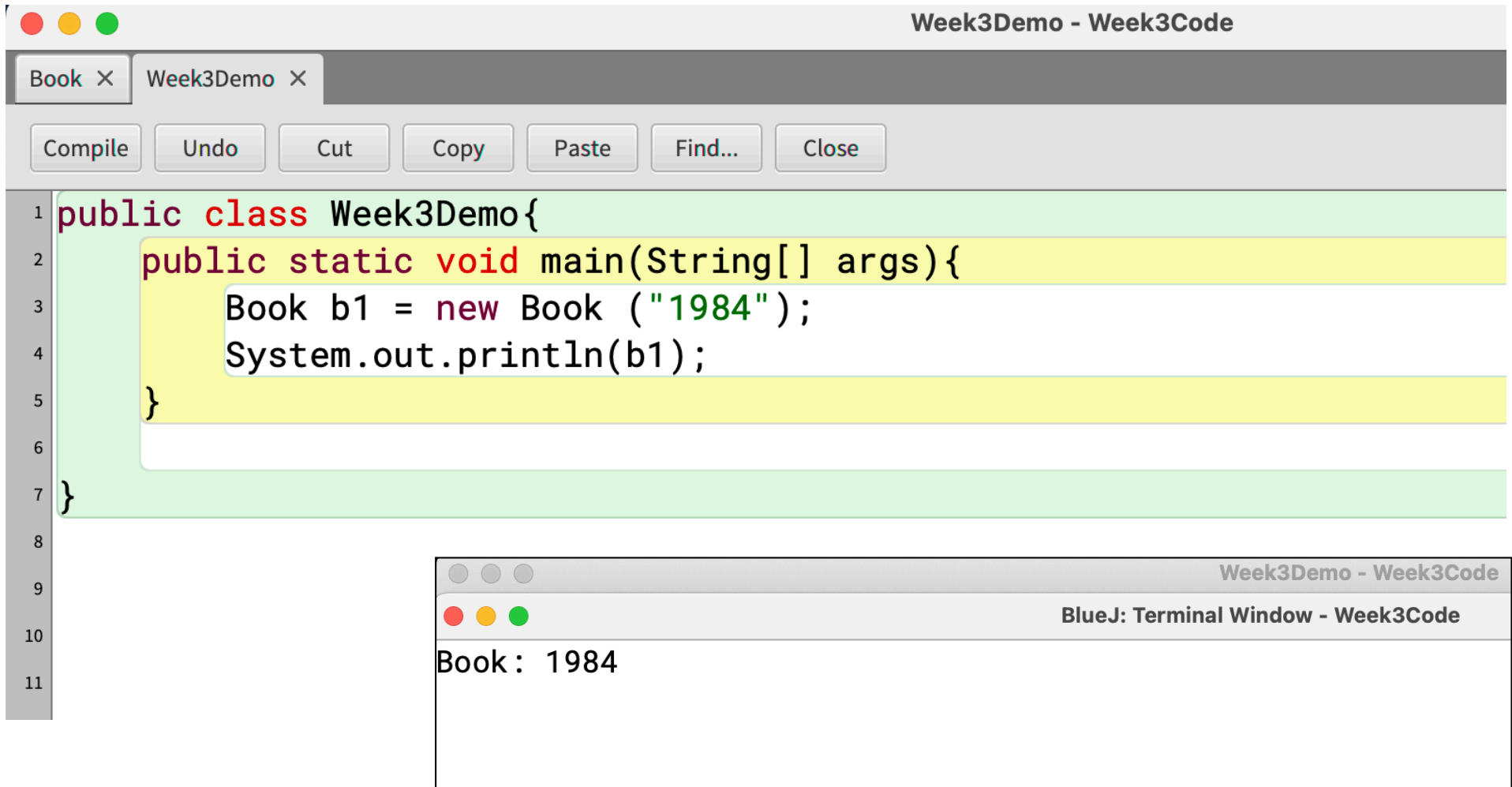


# Book.java

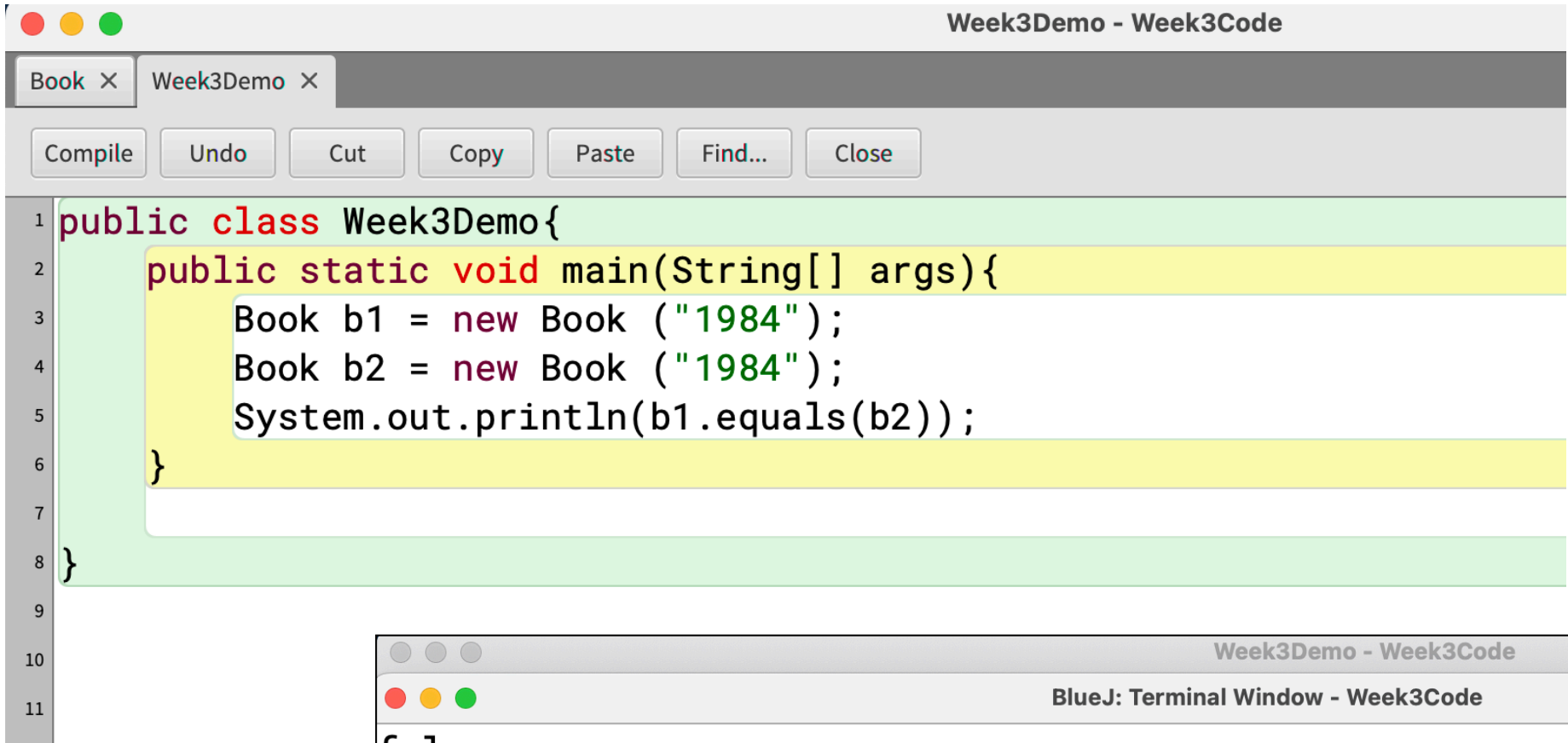


```
1 public class Book{
2     private String title;
3     public Book(String title){
4         this.title = title;
5     }
6     public String getTitle(){
7         return title.toUpperCase();
8     }
9     public String toString(){
10        return "Book: " + getTitle();
11    }
12
13
14
```

# Runner - Book.java



# Runner - Book.java



The image shows a Java IDE window titled "Week3Demo - Week3Code". The code editor contains the following Java code:

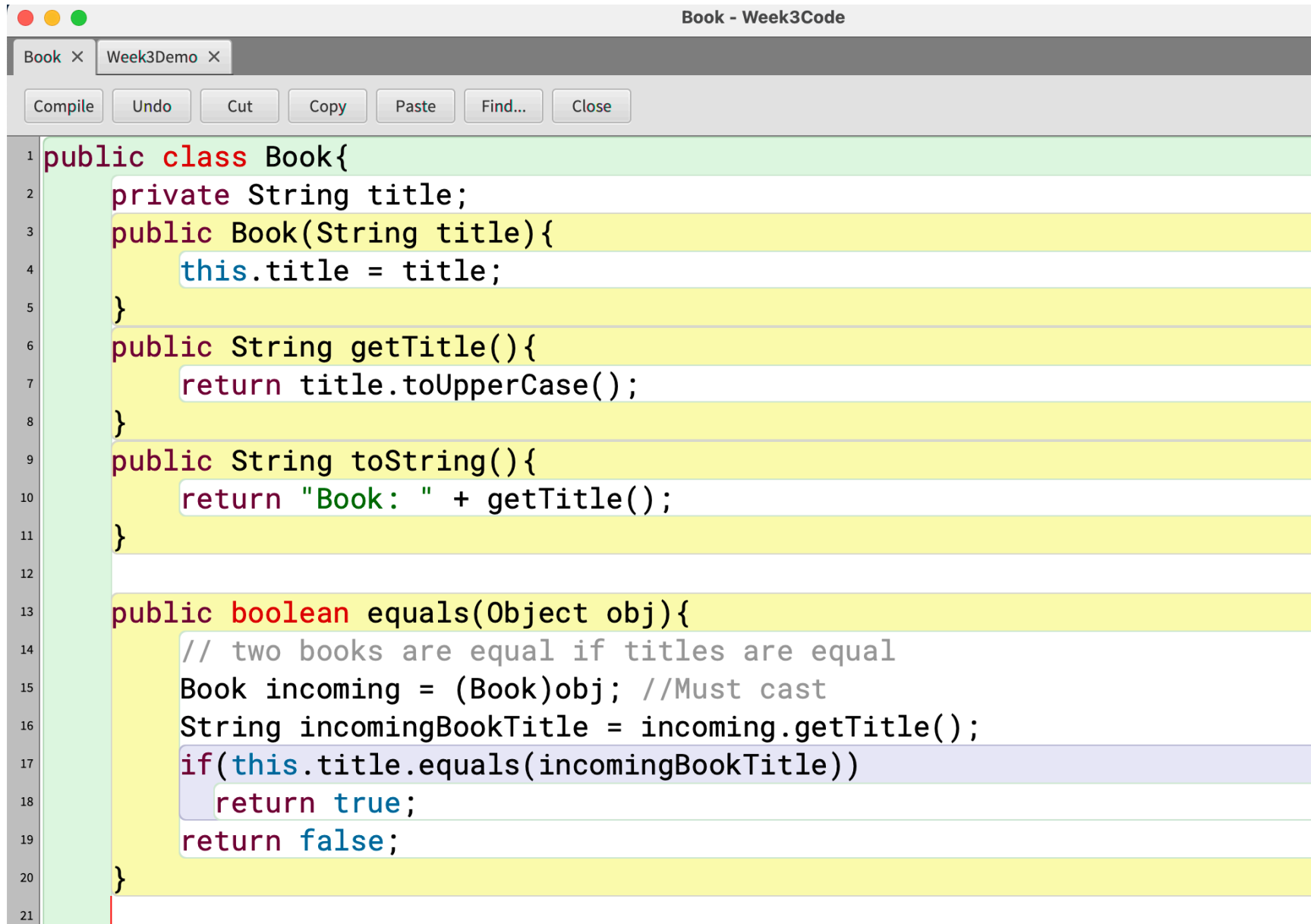
```
1 public class Week3Demo{
2     public static void main(String[] args){
3         Book b1 = new Book ("1984");
4         Book b2 = new Book ("1984");
5         System.out.println(b1.equals(b2));
6     }
7 }
8
9
10
11
```

Below the code editor, there is a terminal window titled "BlueJ: Terminal Window - Week3Code" which displays the output:

```
false
```

Why? Because the Object class provides an equals method. It checks the memory location of the objects and compares those

# Book.java - Custom equals()



The screenshot shows a code editor window titled "Book - Week3Code". The editor has a menu bar with "Book" and "Week3Demo" tabs, and a toolbar with buttons for "Compile", "Undo", "Cut", "Copy", "Paste", "Find...", and "Close". The code is written in Java and is as follows:

```
1 public class Book{
2     private String title;
3     public Book(String title){
4         this.title = title;
5     }
6     public String getTitle(){
7         return title.toUpperCase();
8     }
9     public String toString(){
10        return "Book: " + getTitle();
11    }
12
13    public boolean equals(Object obj){
14        // two books are equal if titles are equal
15        Book incoming = (Book)obj; //Must cast
16        String incomingBookTitle = incoming.getTitle();
17        if(this.title.equals(incomingBookTitle))
18            return true;
19        return false;
20    }
21 }
```

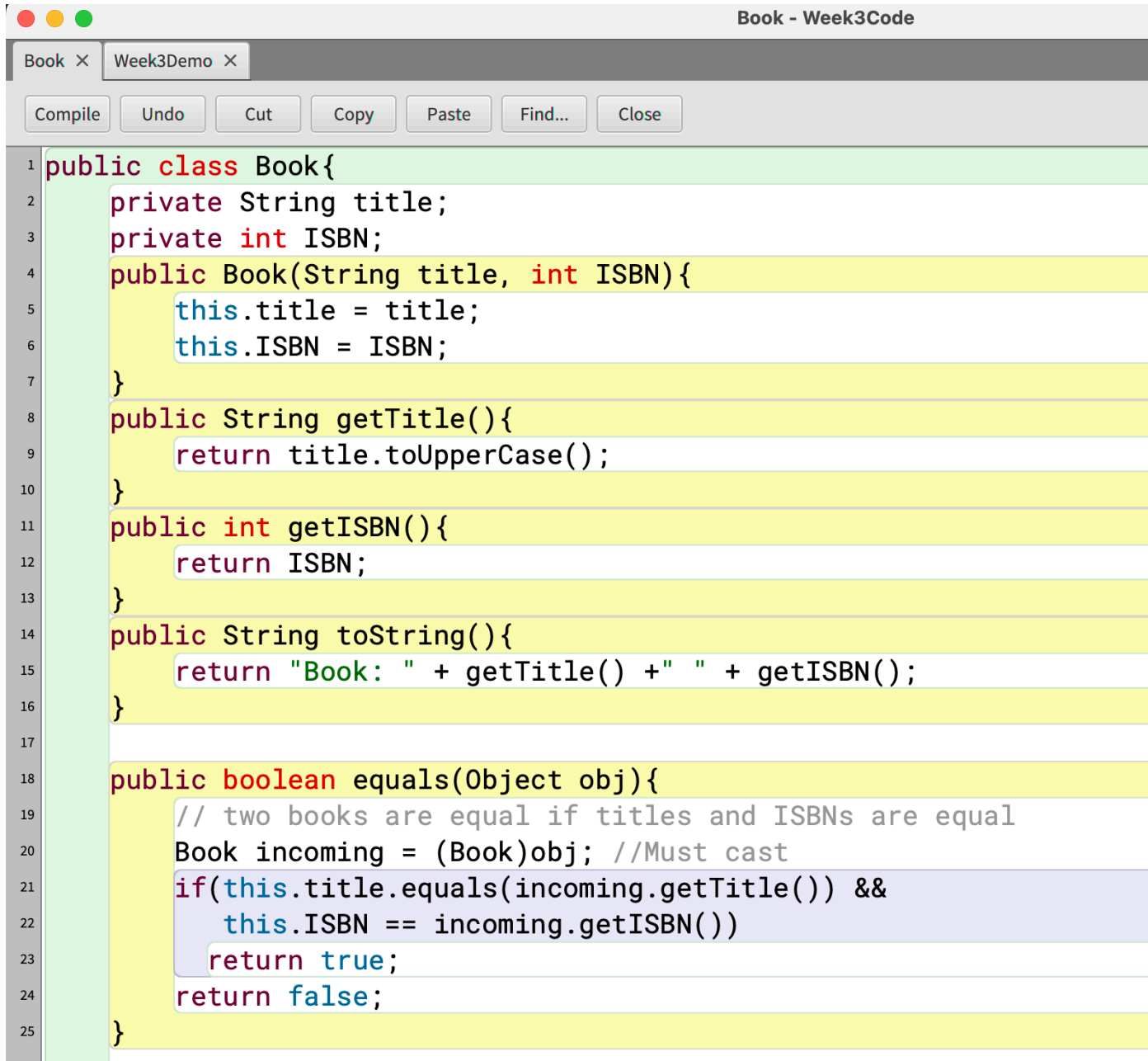
# Book.java - Custom equals() - Runner

The image shows a BlueJ IDE window titled "Week3Demo - Week3Code". It contains a single class file named "Week3Demo". The code defines a public class with a main method that creates two Book objects with the same title "1984" and prints the result of their equals() method. The code is as follows:

```
1 public class Week3Demo{  
2     public static void main(String[] args){  
3         Book b1 = new Book ("1984");  
4         Book b2 = new Book ("1984");  
5         System.out.println(b1.equals(b2));  
6     }  
7 }  
8 }  
9  
10  
11
```

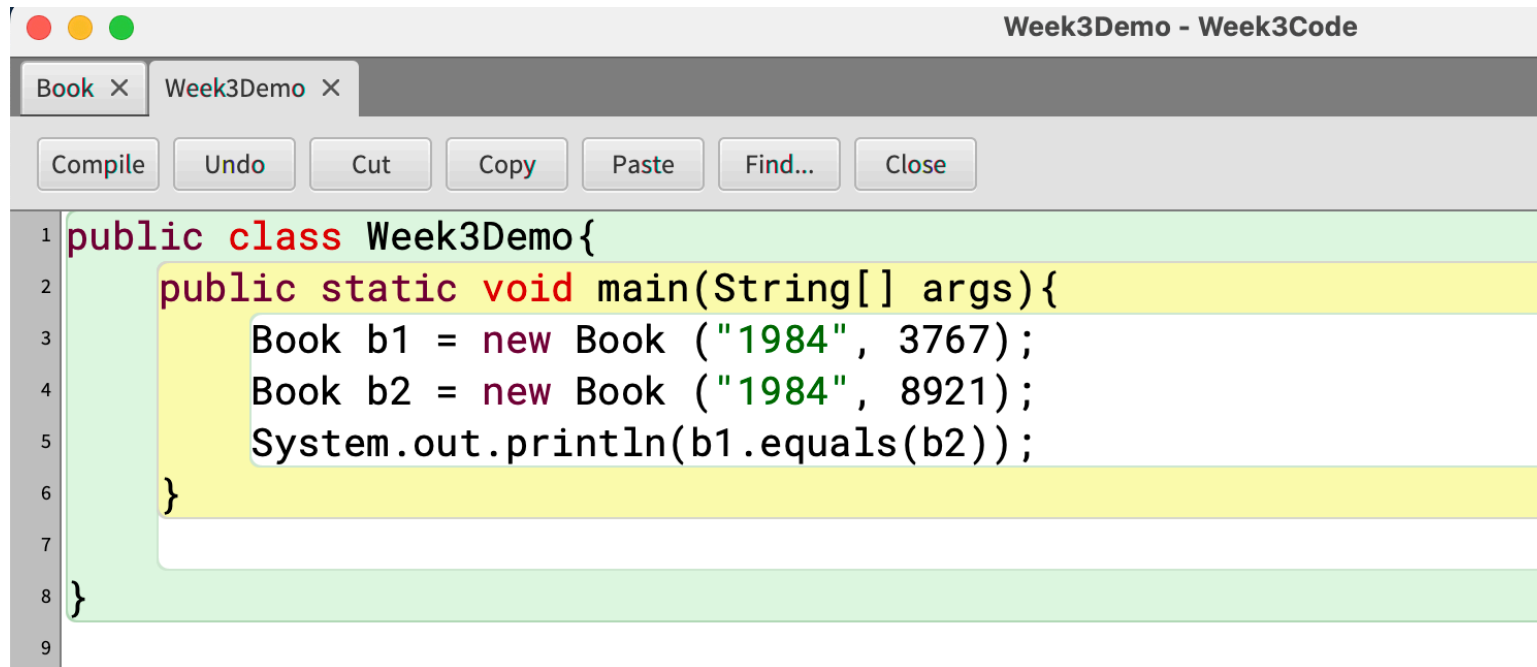
Below the code editor, there is a terminal window titled "Book - Week3Code" with the subtitle "BlueJ: Terminal Window - Week3Code". It displays the output of the program, which is "true".

# Book.java - Custom equals() - using ISBN + Title

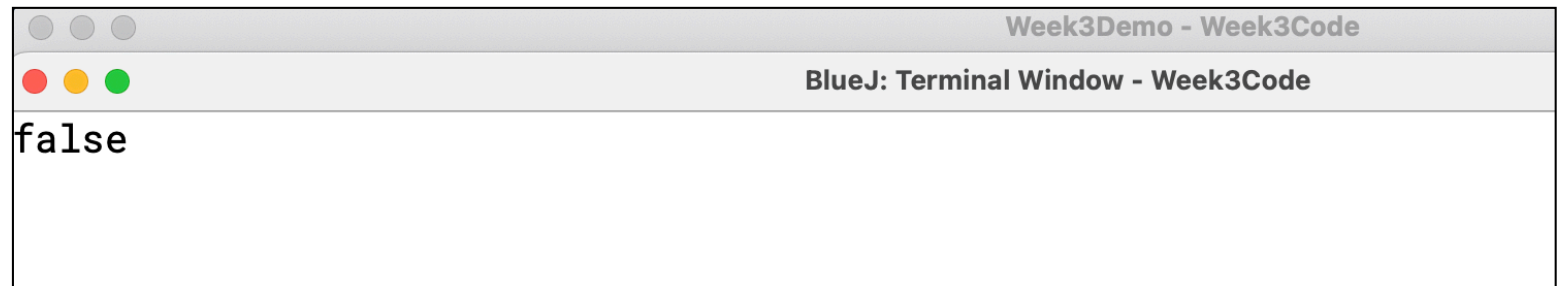


```
1 public class Book{
2     private String title;
3     private int ISBN;
4     public Book(String title, int ISBN){
5         this.title = title;
6         this.ISBN = ISBN;
7     }
8     public String getTitle(){
9         return title.toUpperCase();
10    }
11    public int getISBN(){
12        return ISBN;
13    }
14    public String toString(){
15        return "Book: " + getTitle() + " " + getISBN();
16    }
17
18    public boolean equals(Object obj){
19        // two books are equal if titles and ISBNs are equal
20        Book incoming = (Book)obj; //Must cast
21        if(this.title.equals(incoming.getTitle()) &&
22            this.ISBN == incoming.getISBN())
23            return true;
24        return false;
25    }
}
```

# Book.java - Custom equals() - using ISBN + Title



```
1 public class Week3Demo{
2     public static void main(String[] args){
3         Book b1 = new Book ("1984", 3767);
4         Book b2 = new Book ("1984", 8921);
5         System.out.println(b1.equals(b2));
6     }
7 }
8
9 }
```



```
Week3Demo - Week3Code
BlueJ: Terminal Window - Week3Code
false
```

# Question

---

1. Indicate if the statement is TRUE/FALSE

- (a) Primitive variables' equality is checked with the = symbol
- (b) String equality should always be done with the == operator
- (c) The default equals( ) method checks object state for equality
- (d) An overridden/custom equals( ) method is never needed



# Relationships

Classes, like objects, do not exist in isolation. Very often, an object-oriented program consists of a set of interacting objects whose classes are related in some way.

Relationships between classes are established to either:

- Indicate some sort of sharing between the classes
- Indicate a semantic connection between the classes

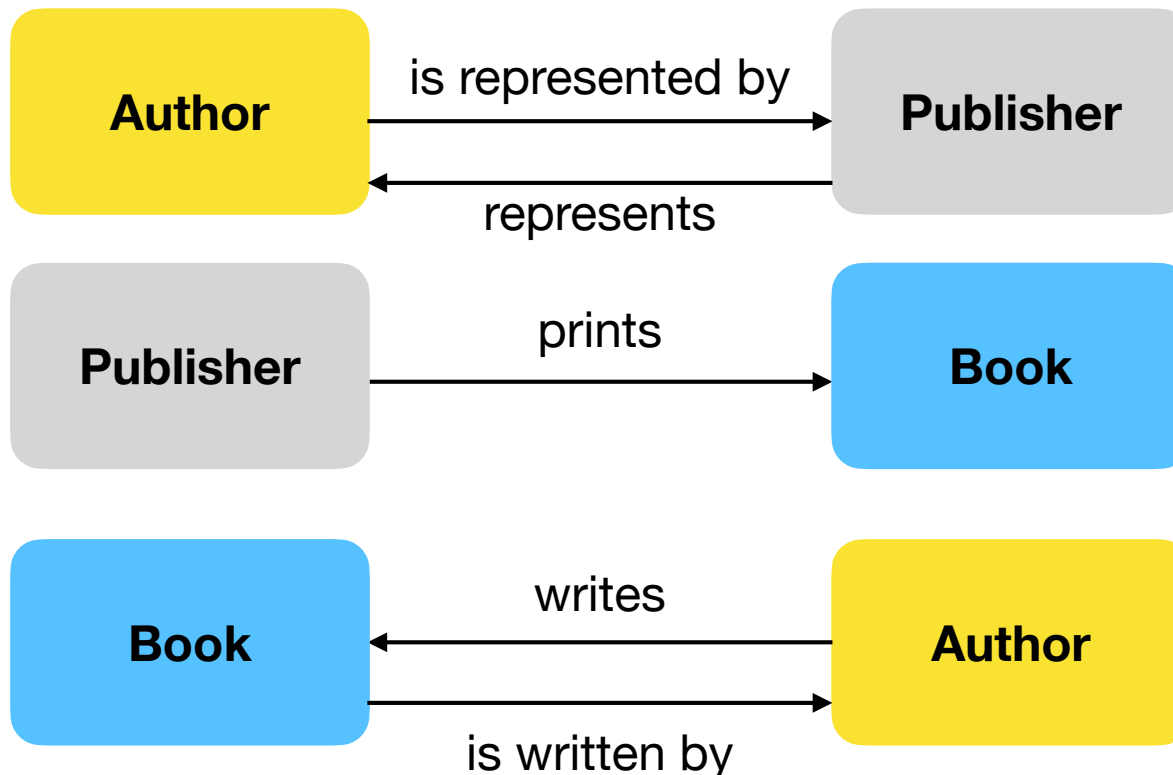
# Kinds of Relationships

There are three basic kinds of relationships:

1. Association/Dependency (uses)
2. Generalisation/Specialisation (is-a)
3. Composition/Aggregation (part-of)

# Associations

An association denotes a semantic dependency between objects. The direction of this association can be bidirectional or can be navigated specifically in one direction.



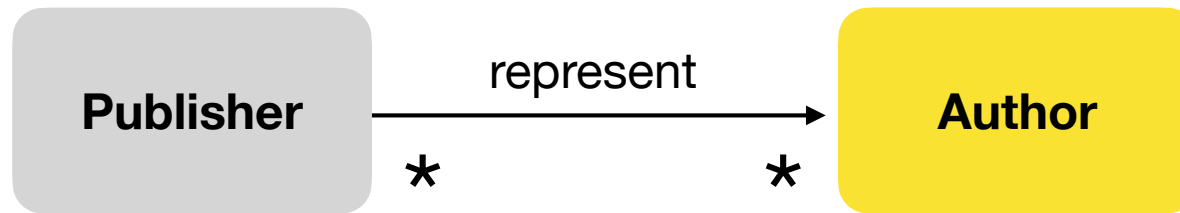
# Cardinality

The cardinality of an association specifies the number of participants in the semantic relationship.

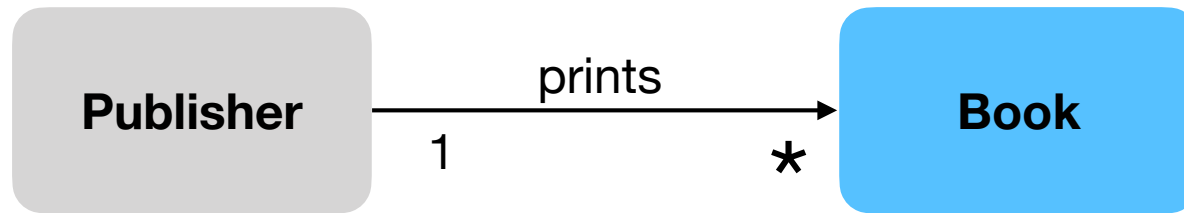
Three common types of cardinality are:

- One-to-one (narrow). (1-1)
- One-to-many (1-\*)
- Many-to-many (\*-\*)

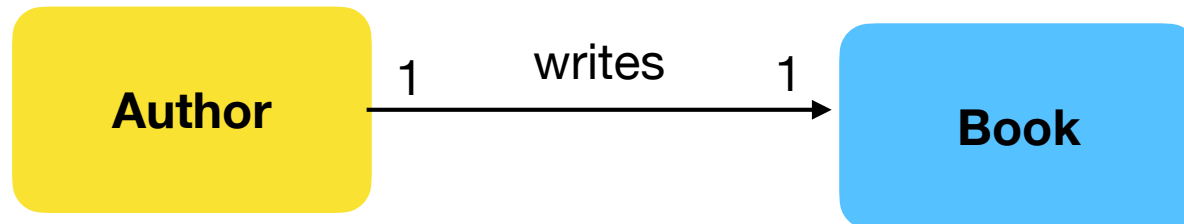
# Example- Associations + Cardinalities



**many-to-many association**

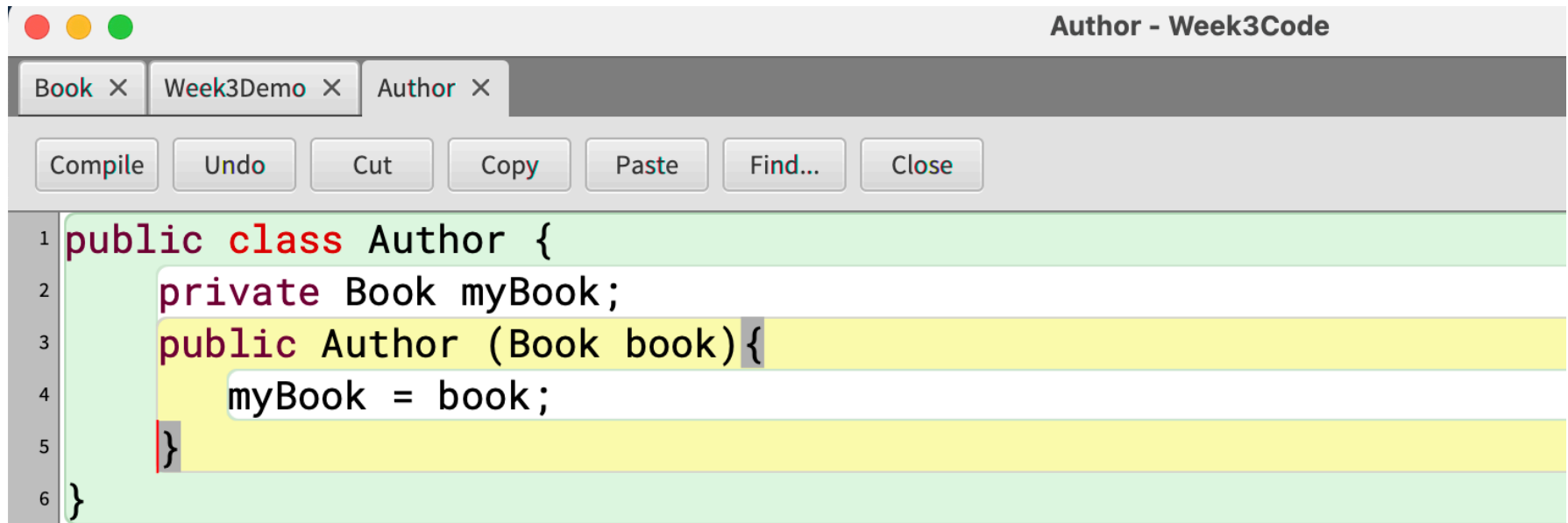


**one-to-many association**

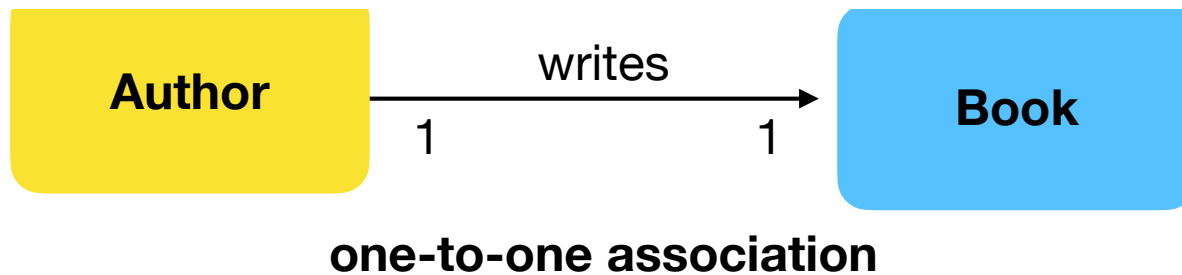


**one-to-one association**

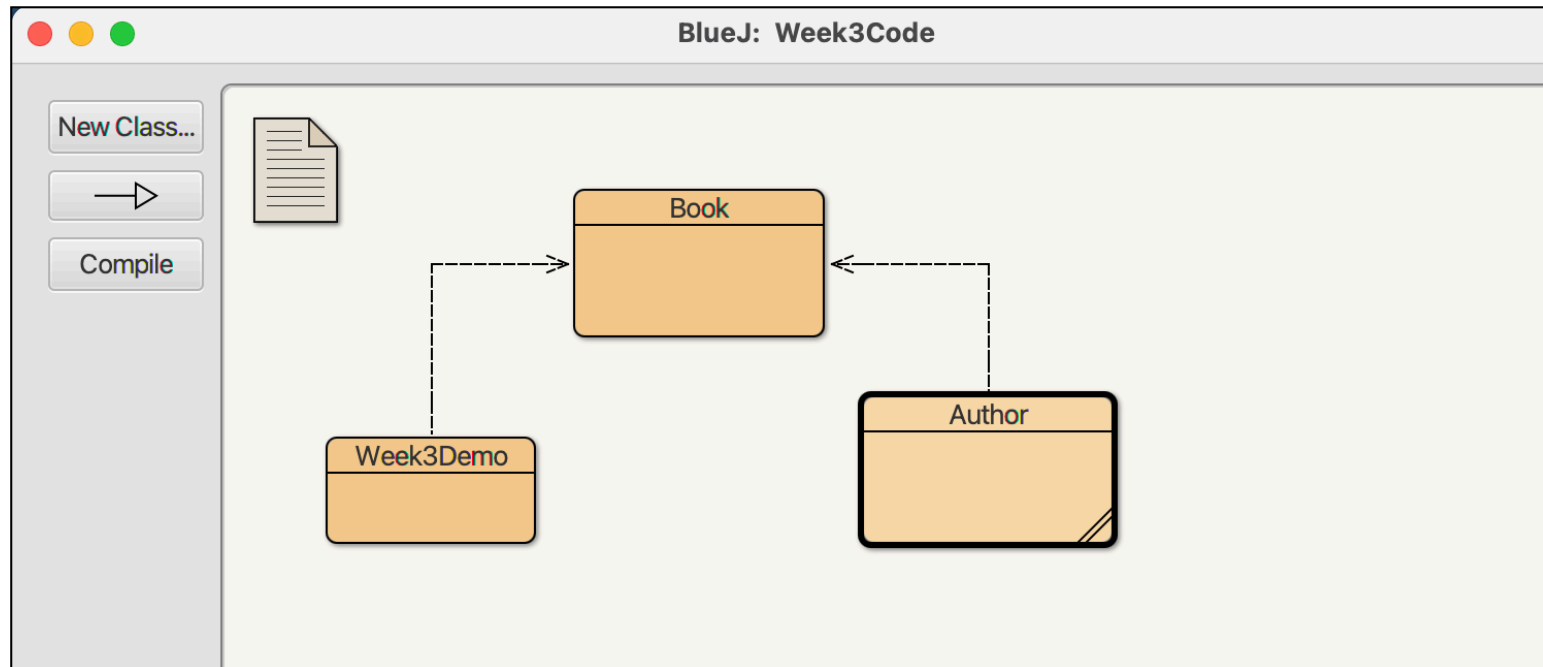
# Example - Implementing Associations 1:1



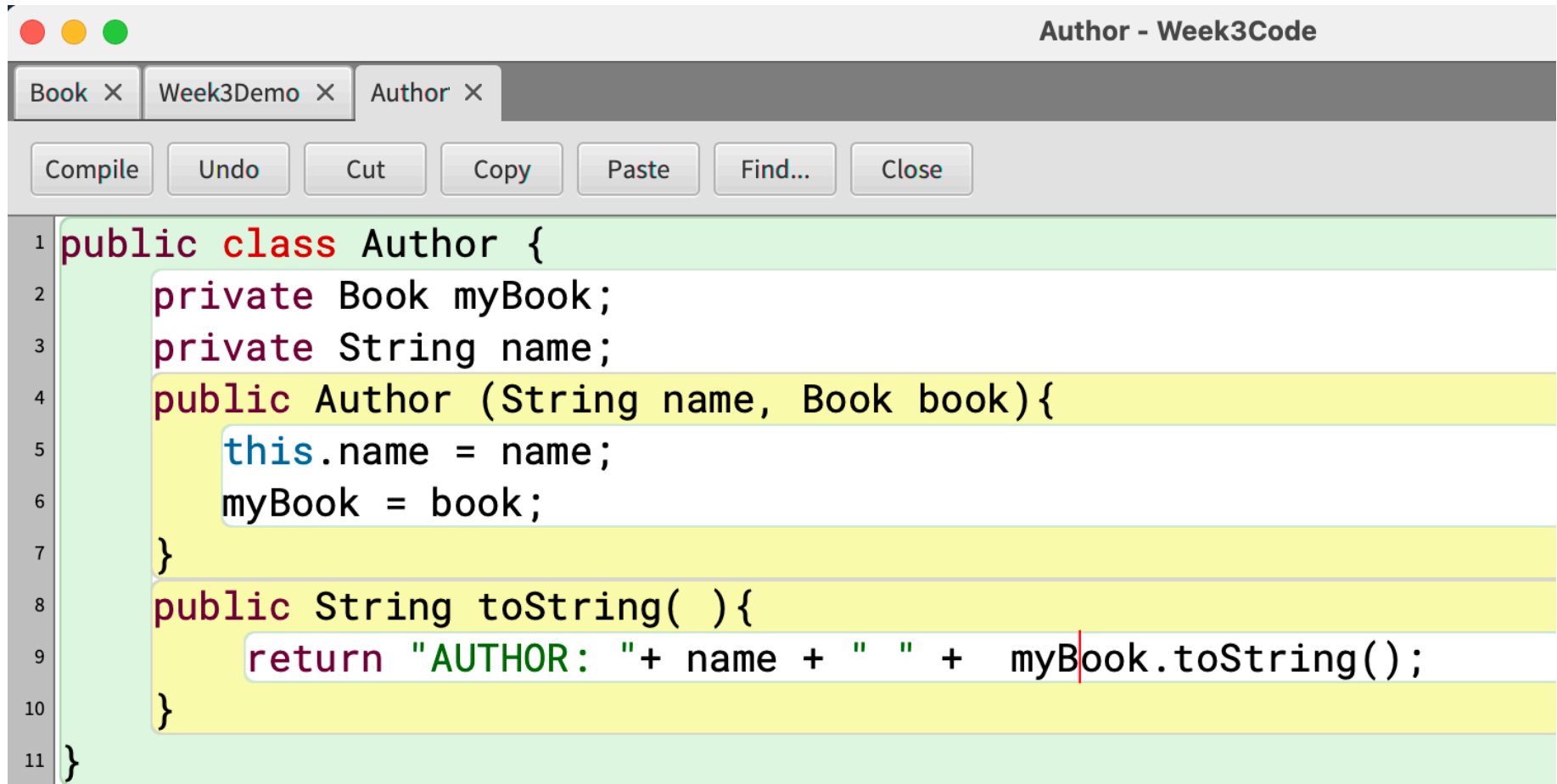
```
1 public class Author {  
2     private Book myBook;  
3     public Author (Book book) {  
4         myBook = book;  
5     }  
6 }
```



# Example - Implementing Associations 1:1



# Example - Implementing Associations 1:1

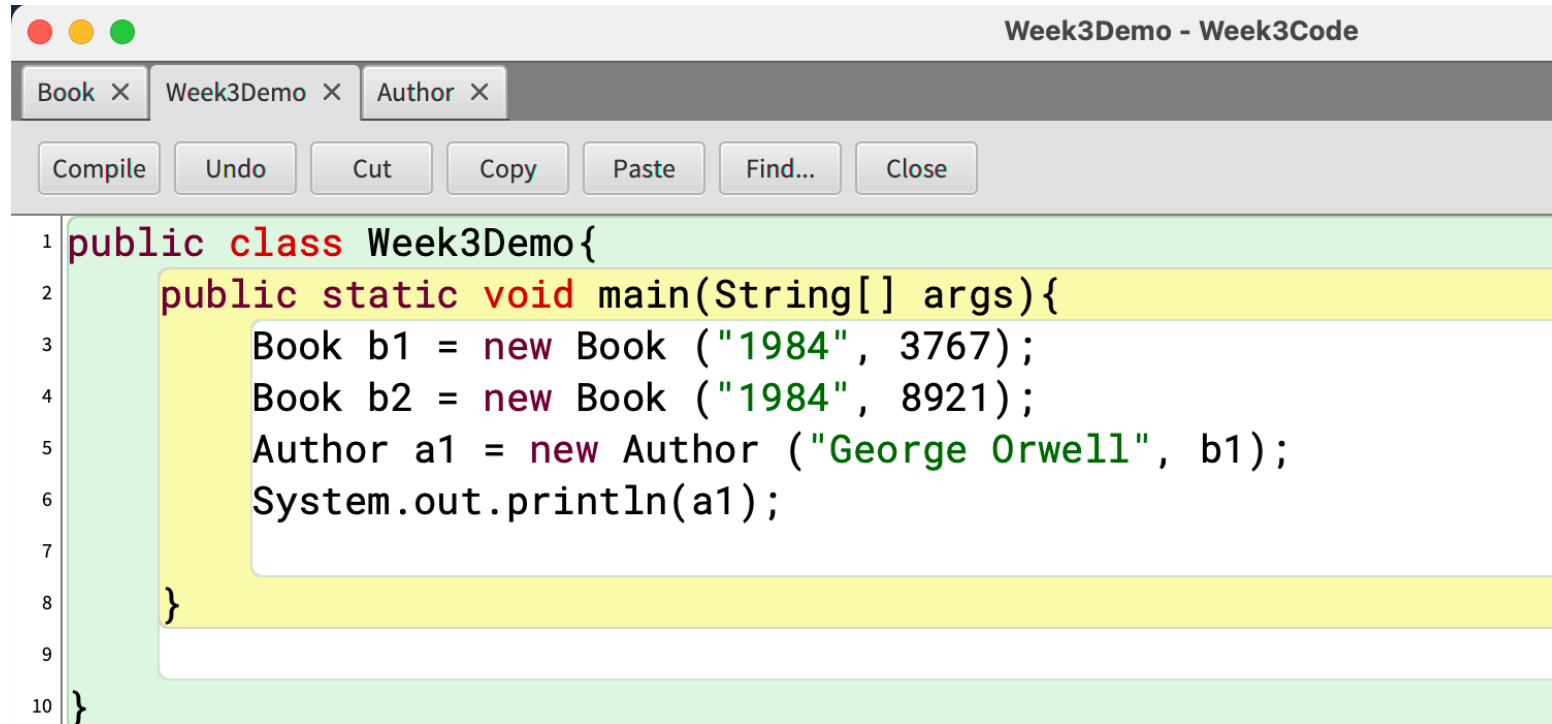


```
1 public class Author {
2     private Book myBook;
3     private String name;
4     public Author (String name, Book book){
5         this.name = name;
6         myBook = book;
7     }
8     public String toString( ){
9         return "AUTHOR: " + name + " " + myBook.toString();
10    }
11 }
```

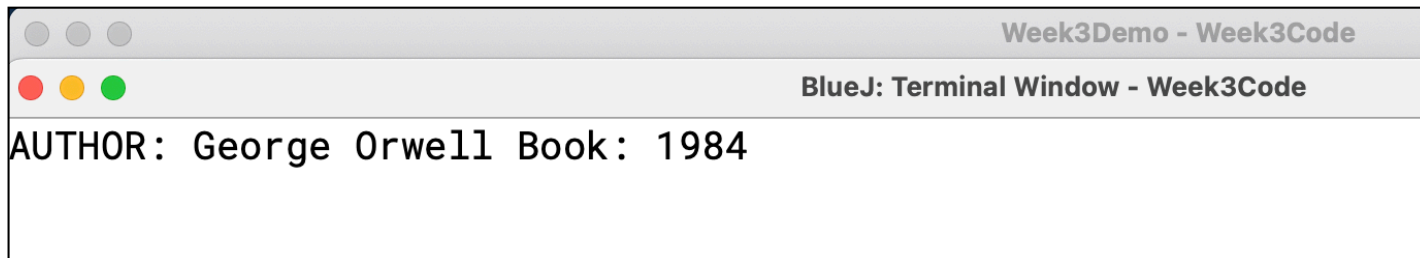


# Example - Implementing Associations

## 1:1



```
1 public class Week3Demo{
2     public static void main(String[] args){
3         Book b1 = new Book ("1984", 3767);
4         Book b2 = new Book ("1984", 8921);
5         Author a1 = new Author ("George Orwell", b1);
6         System.out.println(a1);
7     }
8 }
9
10 }
```



```
AUTHOR: George Orwell Book: 1984
```

# Question

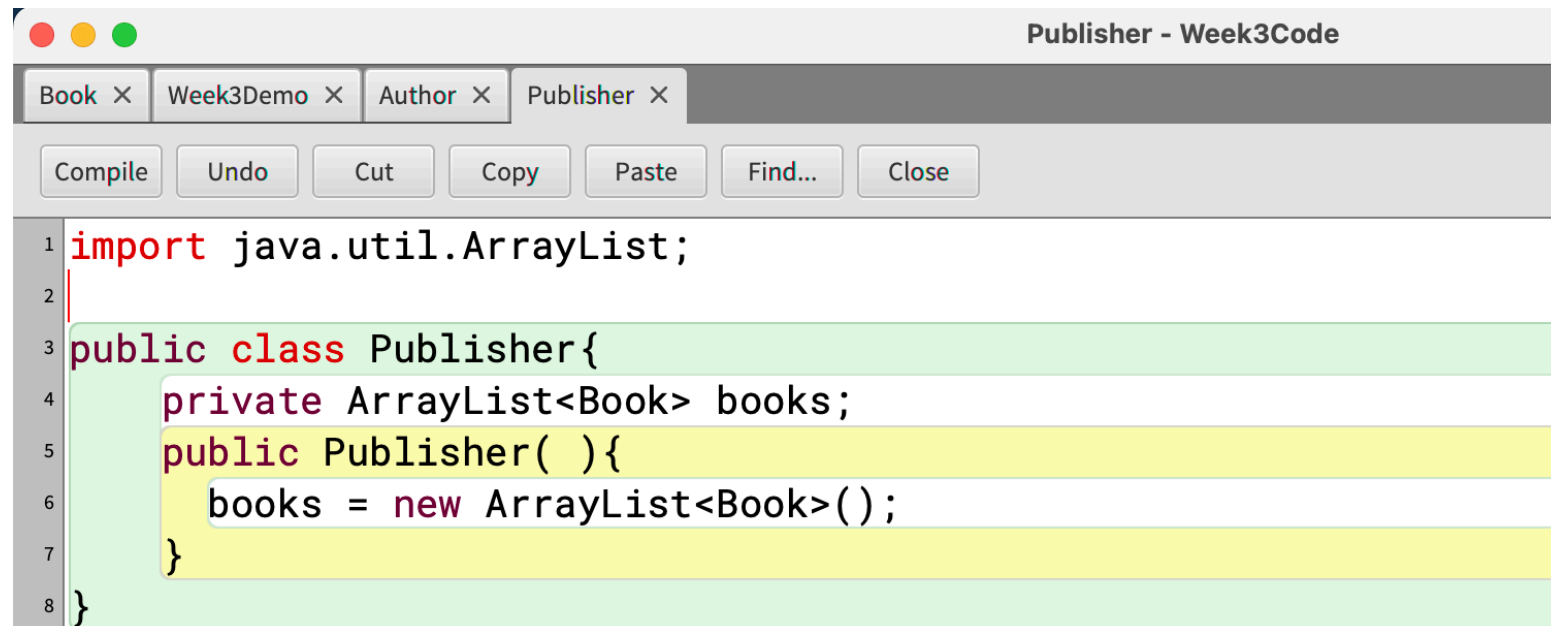
---

1. Indicate if the statement is TRUE/FALSE

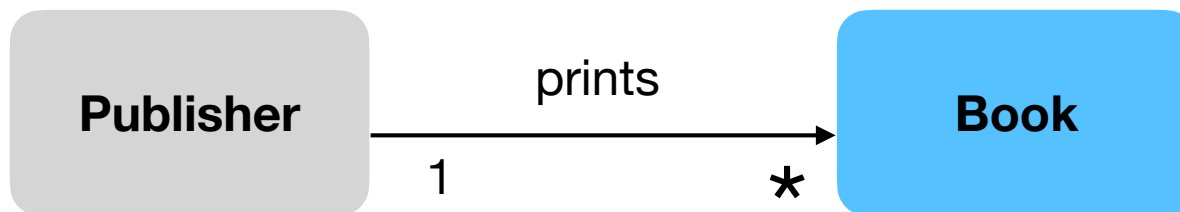
- (a) One to one relationships are always bidirectional
- (b) One to one relationships are implemented with a collection
- (c) One to one relationships are created with object variables
- (d) One to one relationships are dependencies between classes

# Example - Implementing Associations

1: \*



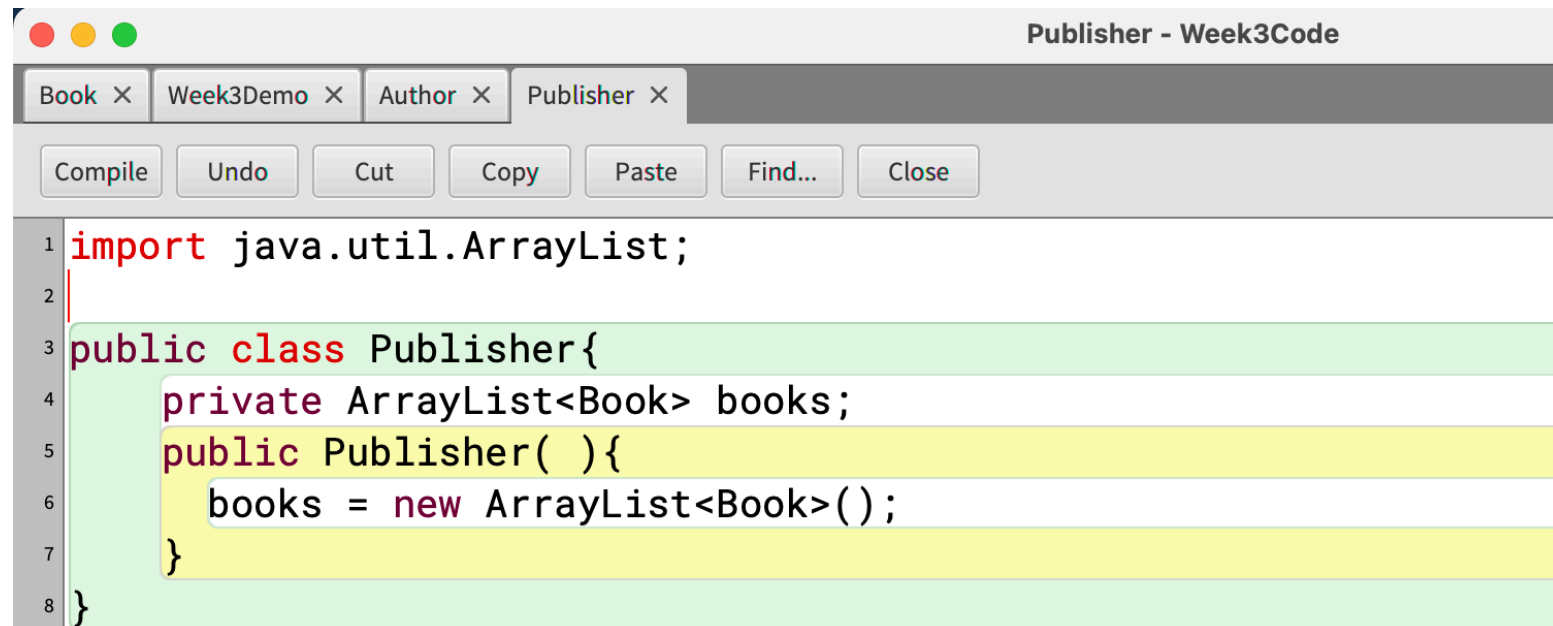
```
1 import java.util.ArrayList;
2
3 public class Publisher{
4     private ArrayList<Book> books;
5     public Publisher( ){
6         books = new ArrayList<Book>();
7     }
8 }
```



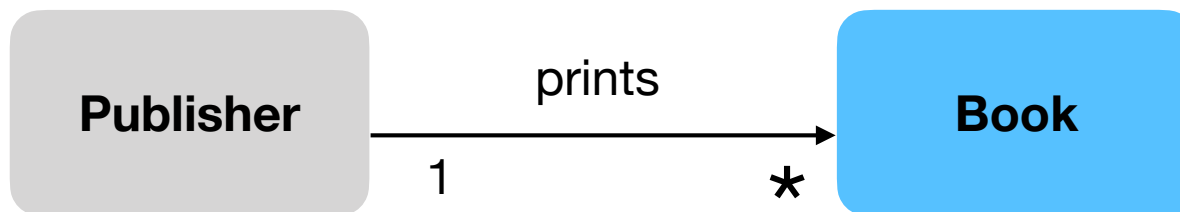
one-to-many association

# Example - Implementing Associations

1: \*



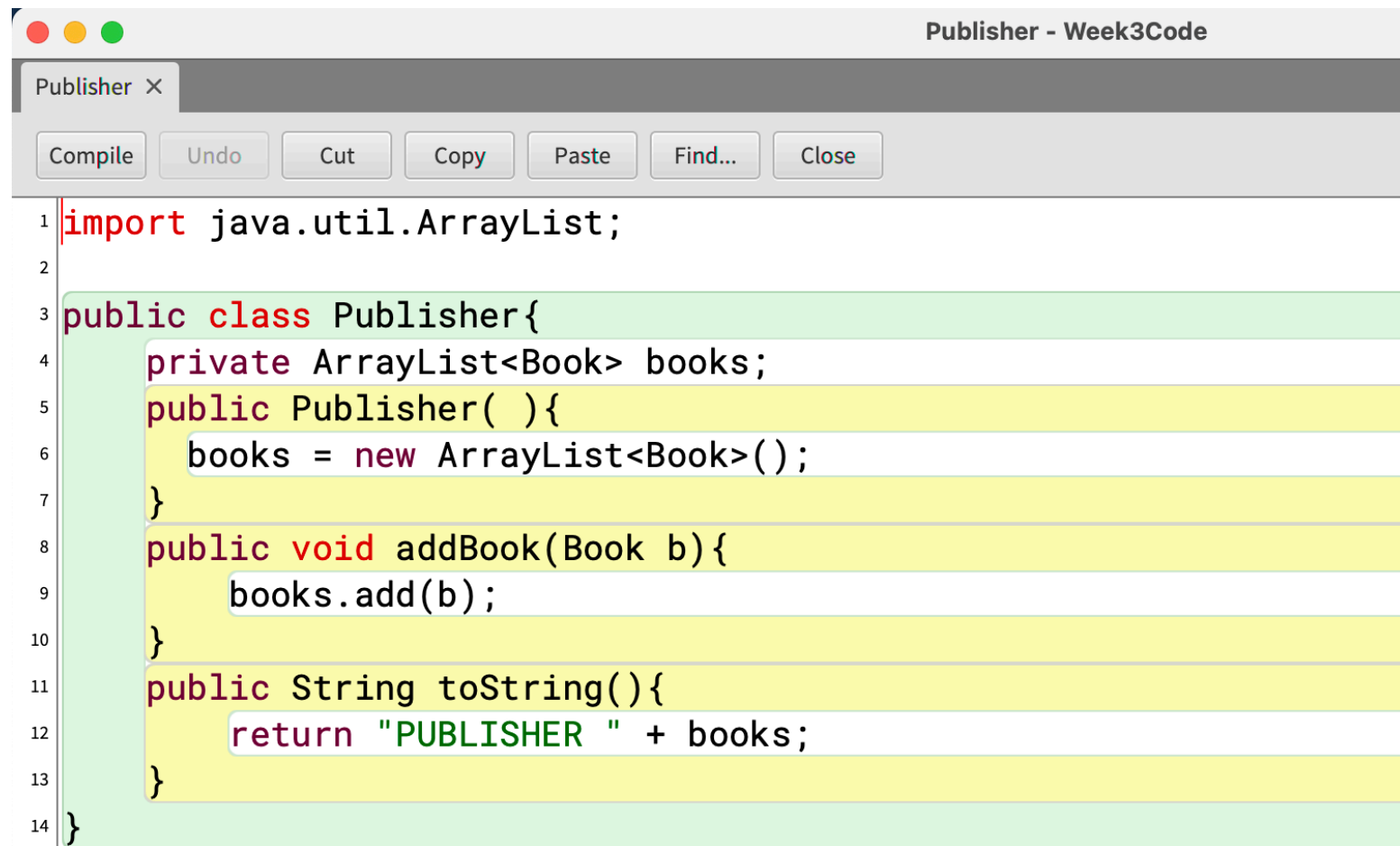
```
1 import java.util.ArrayList;
2
3 public class Publisher{
4     private ArrayList<Book> books;
5     public Publisher( ){
6         books = new ArrayList<Book>();
7     }
8 }
```



one-to-many association

# Example - Implementing Associations

1: \*



```
1 import java.util.ArrayList;
2
3 public class Publisher{
4     private ArrayList<Book> books;
5     public Publisher( ){
6         books = new ArrayList<Book>();
7     }
8     public void addBook(Book b){
9         books.add(b);
10    }
11    public String toString(){
12        return "PUBLISHER " + books;
13    }
14 }
```

# Example - Implementing Associations

1: \*

The screenshot displays the BlueJ IDE interface. The main window, titled 'Week3Demo - Week3Code', contains a Java class named 'Week3Demo'. The code defines a 'main' method that creates two 'Book' objects ('1984' and 'Dune'), two 'Author' objects ('George Orwell' and 'Frank Herbert'), and a 'Publisher' object. The 'Publisher' object is then populated with the two 'Book' objects, and the result is printed to the console. Below the code editor, a smaller window titled 'Publisher - Week3Code' shows the output of the program, which is 'PUBLISHER [Book: 1984, Book: DUNE]'. The code is color-coded: keywords in red, strings in green, and identifiers in black. The 'main' method body is highlighted in yellow.

```
1 public class Week3Demo{
2     public static void main(String[] args){
3         Book b1 = new Book ("1984", 3767);
4         Book b2 = new Book ("Dune", 8949);
5         Author a1 = new Author ("George Orwell", b1);
6         Author a2 = new Author ("Frank Herbert", b2);
7         Publisher p = new Publisher();
8         p.addBook(b1);
9         p.addBook(b2);
10        System.out.println(p);
11    }
12
13 }
```

PUBLISHER [Book: 1984, Book: DUNE]

# Example - Implementing Associations

1: \*

The screenshot displays the BlueJ IDE interface. The main window, titled 'Week3Demo - Week3Code', contains a code editor with the following Java code:

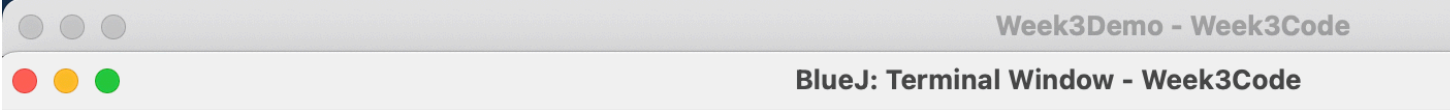
```
1 public class Week3Demo{
2     public static void main(String[] args){
3         Book b1 = new Book ("1984", 3767);
4         Book b2 = new Book ("Dune", 8949);
5         Author a1 = new Author ("George Orwell", b1);
6         Author a2 = new Author ("Frank Herbert", b2);
7         Publisher p = new Publisher();
8         p.addBook(b1);
9         p.addBook(b2);
10        System.out.println(p);
11    }
12 }
13 }
```

Below the code editor, a terminal window titled 'Publisher - Week3Code' shows the output of the program:

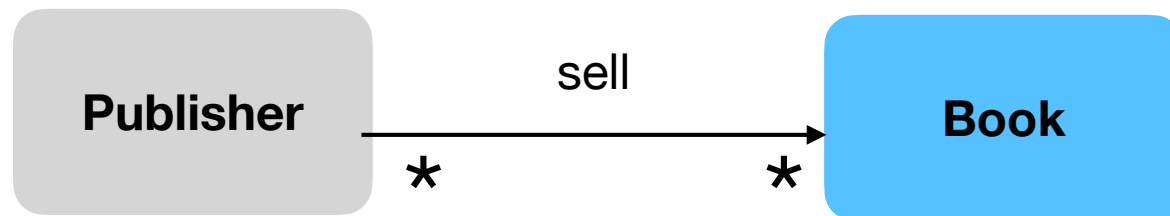
```
BlueJ: Terminal Window - Week3Code
PUBLISHER [Book: 1984, Book: DUNE]
```

# Example - Implementing Associations

\*.\*  
.



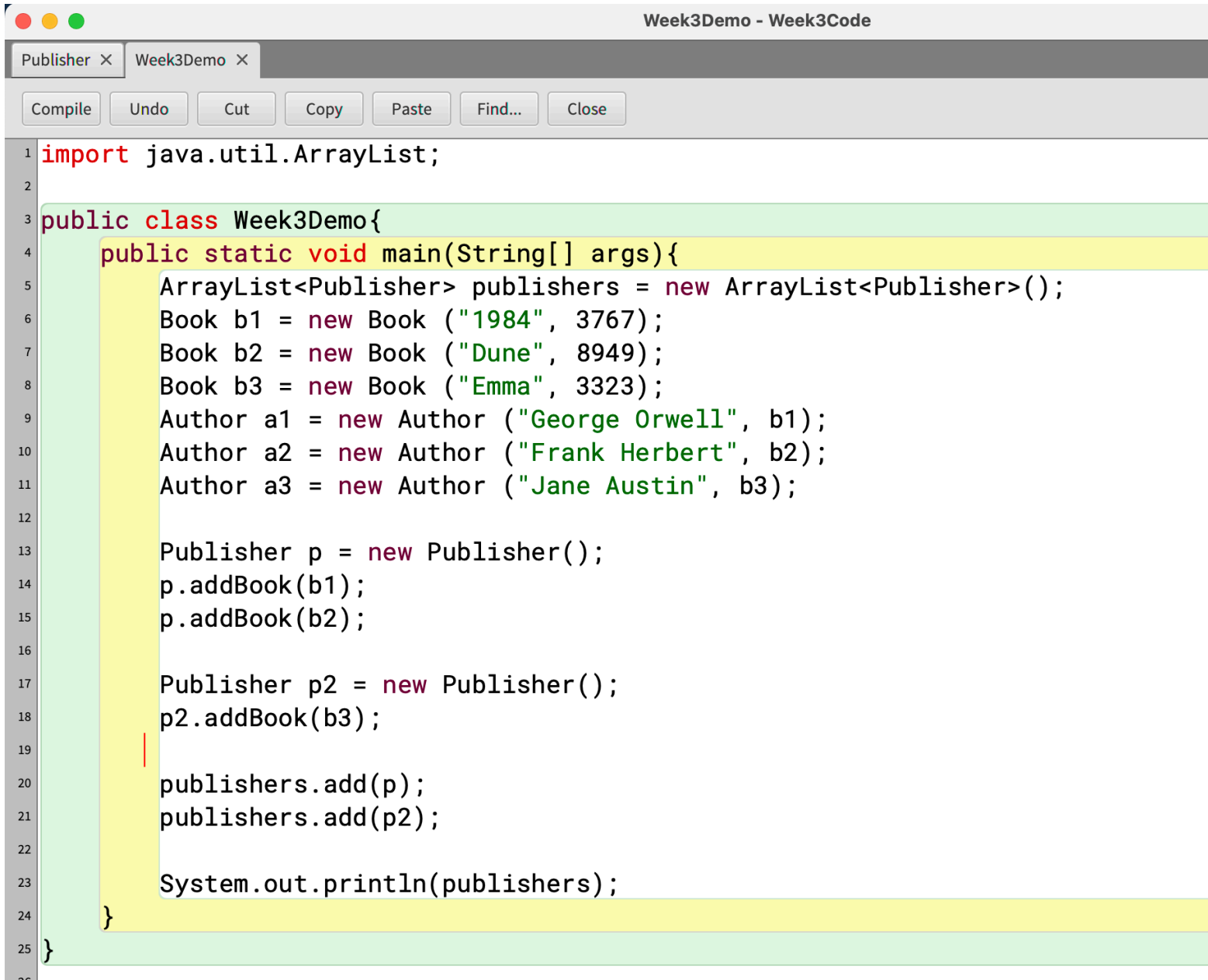
```
[PUBLISHER [Book: 1984, Book: DUNE], PUBLISHER [Book: EMMA]]
```





# Example - Implementing Associations

\*.\*  
■



```
1 import java.util.ArrayList;
2
3 public class Week3Demo{
4     public static void main(String[] args){
5         ArrayList<Publisher> publishers = new ArrayList<Publisher>();
6         Book b1 = new Book ("1984", 3767);
7         Book b2 = new Book ("Dune", 8949);
8         Book b3 = new Book ("Emma", 3323);
9         Author a1 = new Author ("George Orwell", b1);
10        Author a2 = new Author ("Frank Herbert", b2);
11        Author a3 = new Author ("Jane Austin", b3);
12
13        Publisher p = new Publisher();
14        p.addBook(b1);
15        p.addBook(b2);
16
17        Publisher p2 = new Publisher();
18        p2.addBook(b3);
19
20        publishers.add(p);
21        publishers.add(p2);
22
23        System.out.println(publishers);
24    }
25 }
```

# String Class

The String class is an immutable class. It has no mutators and it is impossible to change the state of the class after it has been created.

Several methods are available for use when you create a String object.

```
String s = "chickens";  
boolean plural = s.endsWith("s");
```

<https://docs.oracle.com/javase/7/docs/api/java/lang/String.html>

# Summary

Today you learned about:

- Variable assignment vs equality
- Object equality and the equals( )
- String equality
- Types of Relationships in Object-Oriented Programming
  - Associations
    - 1:1
    - 1:Many
    - Many:Many
  - ArrayLists



# References

- Booch, G. (2007) Object-Oriented Analysis and Design. Chapter 2 - the Object Model
- Chapter 2 Objects: Using, Creating, and Defining:  
<https://runestone.academy/ns/books/published/javajavajava/chapter-objects.html>
- Chapter 3 Methods: Communicating With Objects:  
<https://runestone.academy/ns/books/published/javajavajava/chapter-methods.html>