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

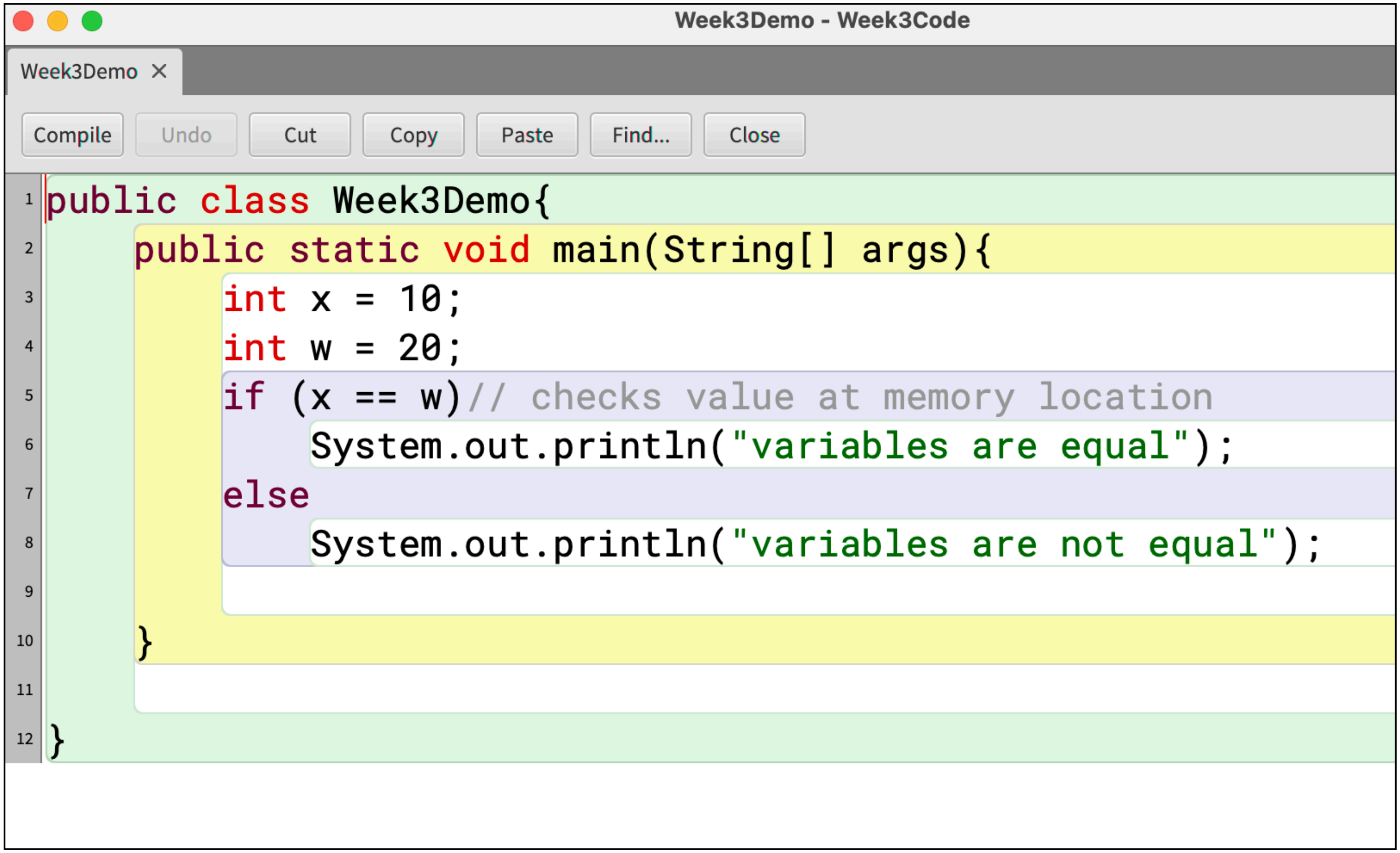
Variable Assignment vs Equality

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

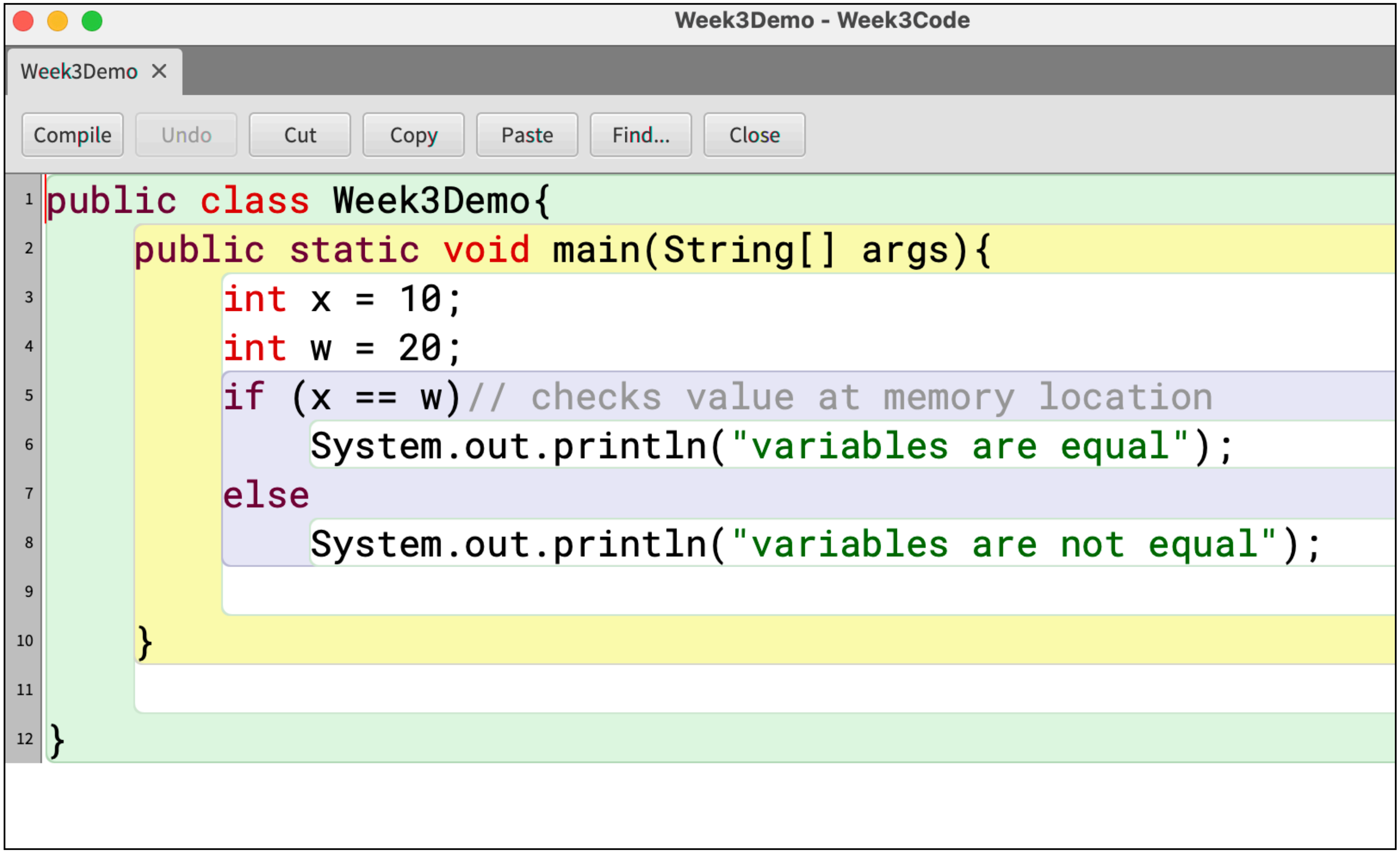
Variable Equality - Example 1



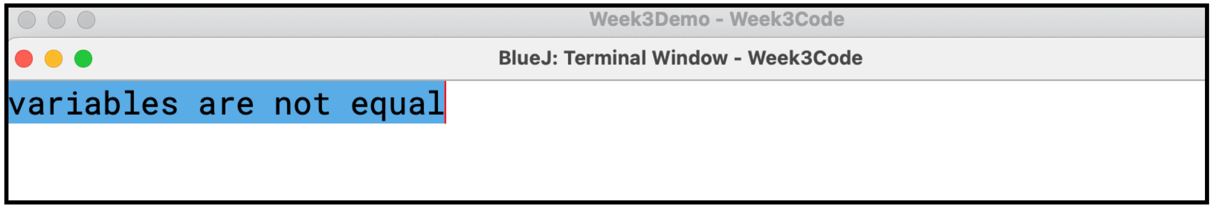
```java
public class Week3Demo{
        public static void main(String[] args){
                int x = 10;
                int w = 20;
                if (x == w)// checks value at memory location
                        System.out.println("variables are equal");
                else
                        System.out.println("variables are not equal");

        }

}
```

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

Variable Equality - Example 1

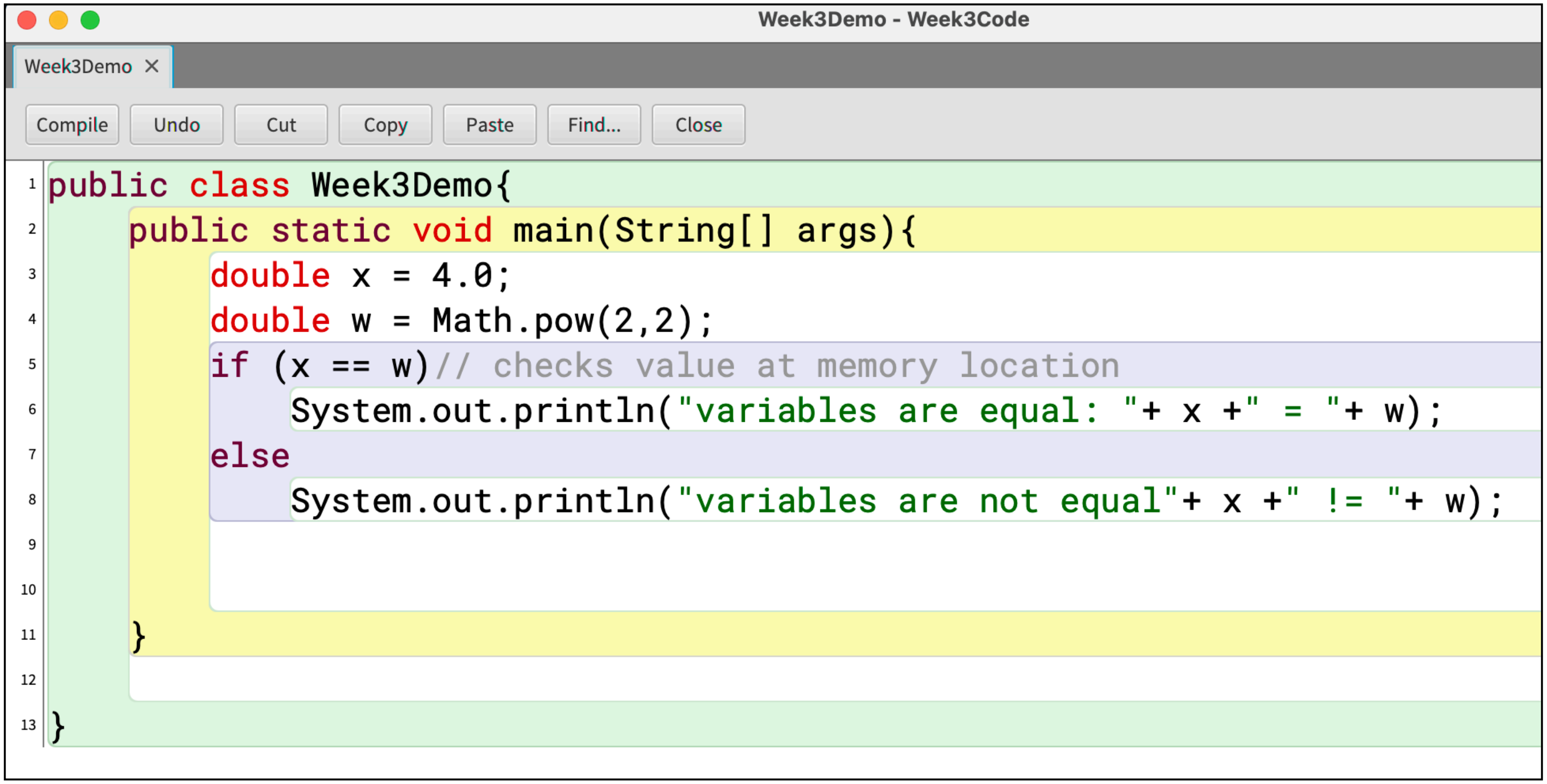


```
public class Week3Demo{
    public static void main(String[] args){
        int x = 10;
        int w = 20;
        if (x == w)// checks value at memory location
            System.out.println("variables are equal");
        else
            System.out.println("variables are not equal");

    }

}
```



```
variables are not equal
```

Variable Equality - Example 2



```java
public class Week3Demo{
    public static void main(String[] args){
        double x = 4.0;
        double w = Math.pow(2,2);
        if (x == w)// checks value at memory location
            System.out.println("variables are equal: "+ x +" = "+ w);
        else
            System.out.println("variables are not equal"+ x +" != "+ w);


    }

}
```

What is printed?

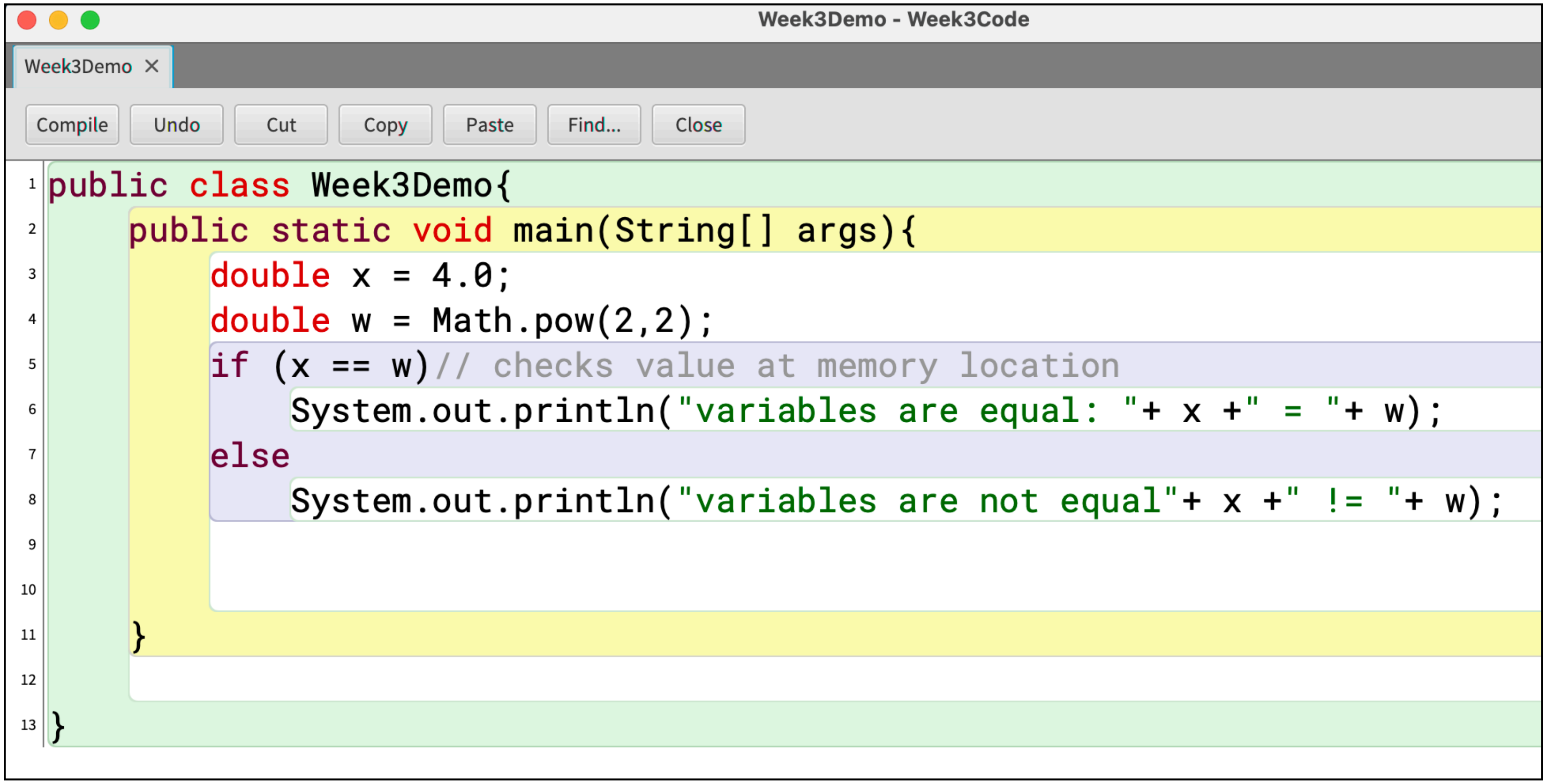
(A) variables are not equal 4.0 != 2

(B) variables are not equal 4.0 != 4

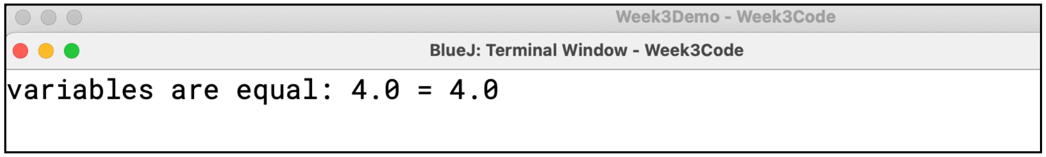
(C) variables are equal 2 = 2

(D) variables are equal 4.0 = 4.0

Variable Equality - Example 2



```
public class Week3Demo{
    public static void main(String[] args){
        double x = 4.0;
        double w = Math.pow(2,2);
        if (x == w)// checks value at memory location
            System.out.println("variables are equal: "+ x +" = "+ w);
        else
            System.out.println("variables are not equal"+ x +" != "+ w);


    }

}
```



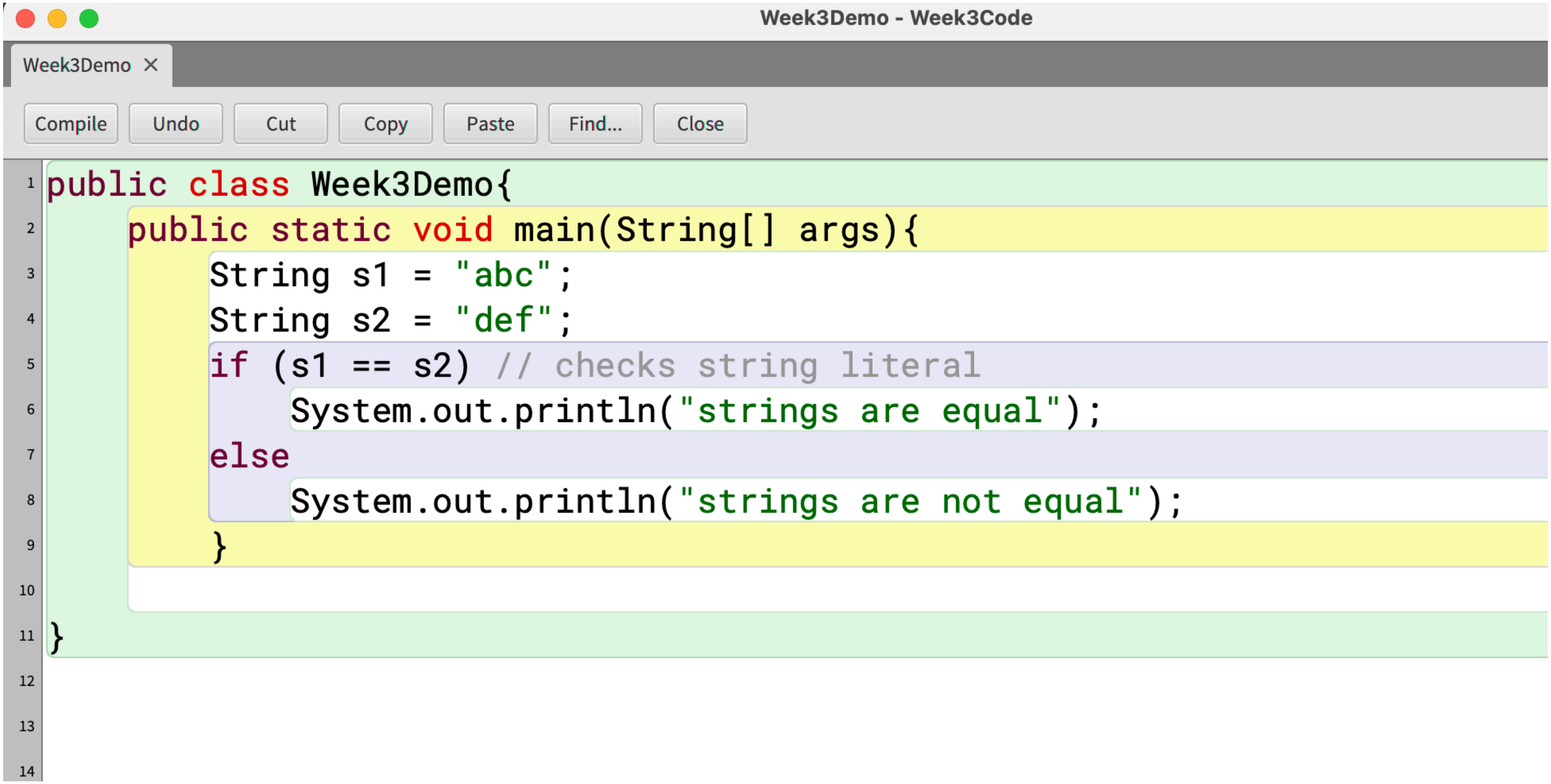
```
variables are equal: 4.0 = 4.0
```

String Equality

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

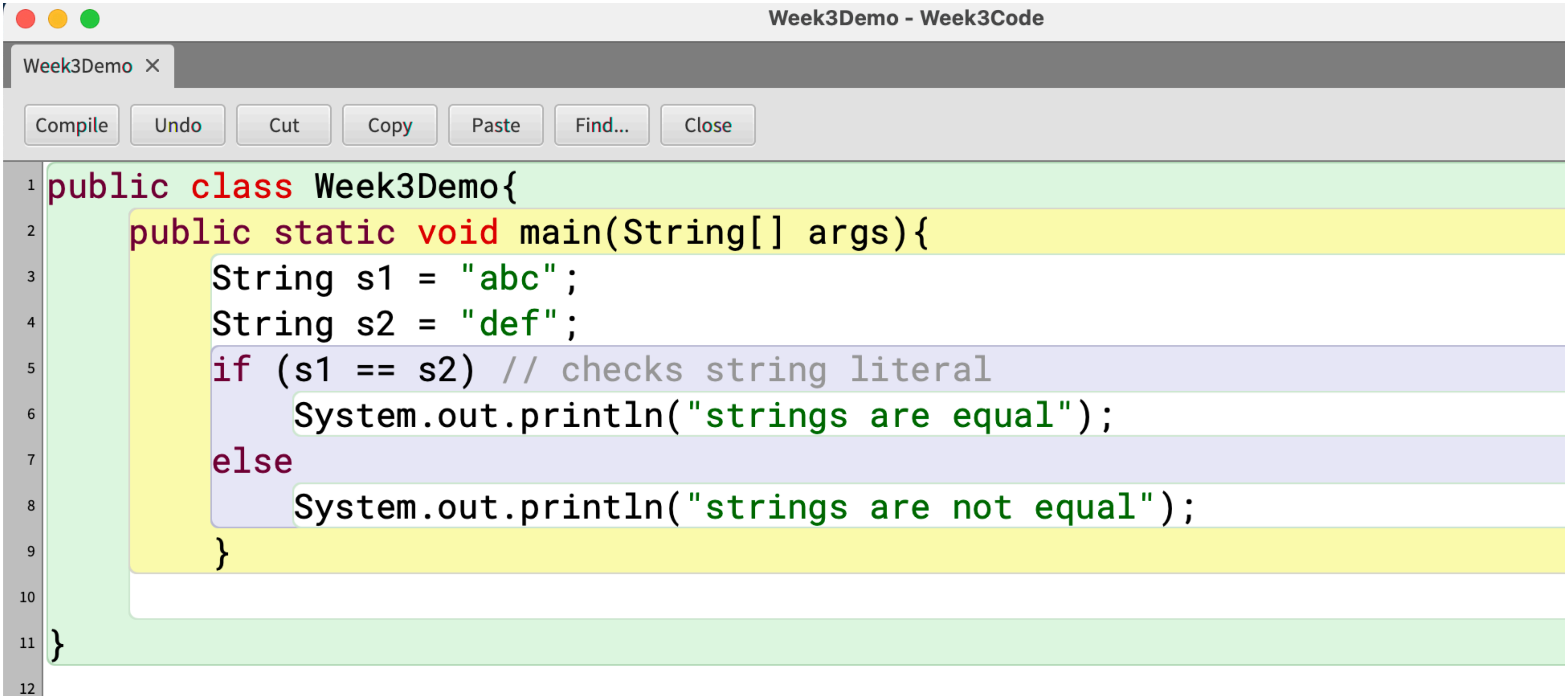
https://www.baeldung.com/java-compare-strings

String Equality - Example 1

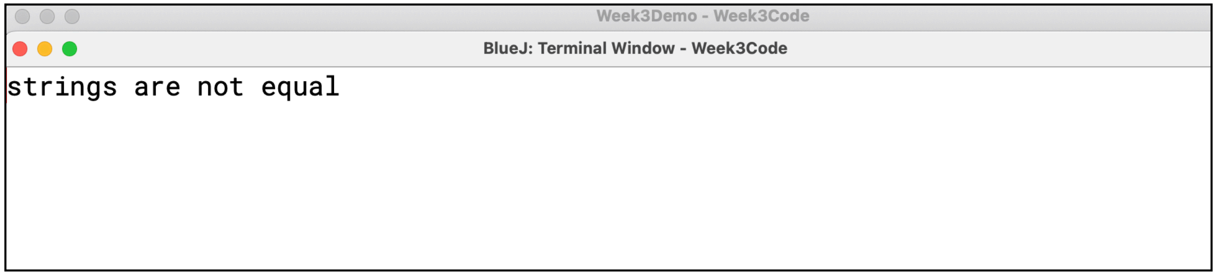


```
public class Week3Demo{
    public static void main(String[] args){
        String s1 = "abc";
        String s2 = "def";
        if (s1 == s2) // checks string literal
            System.out.println("strings are equal");
        else
            System.out.println("strings are not equal");
        }

}
```

String Equality - Example 1

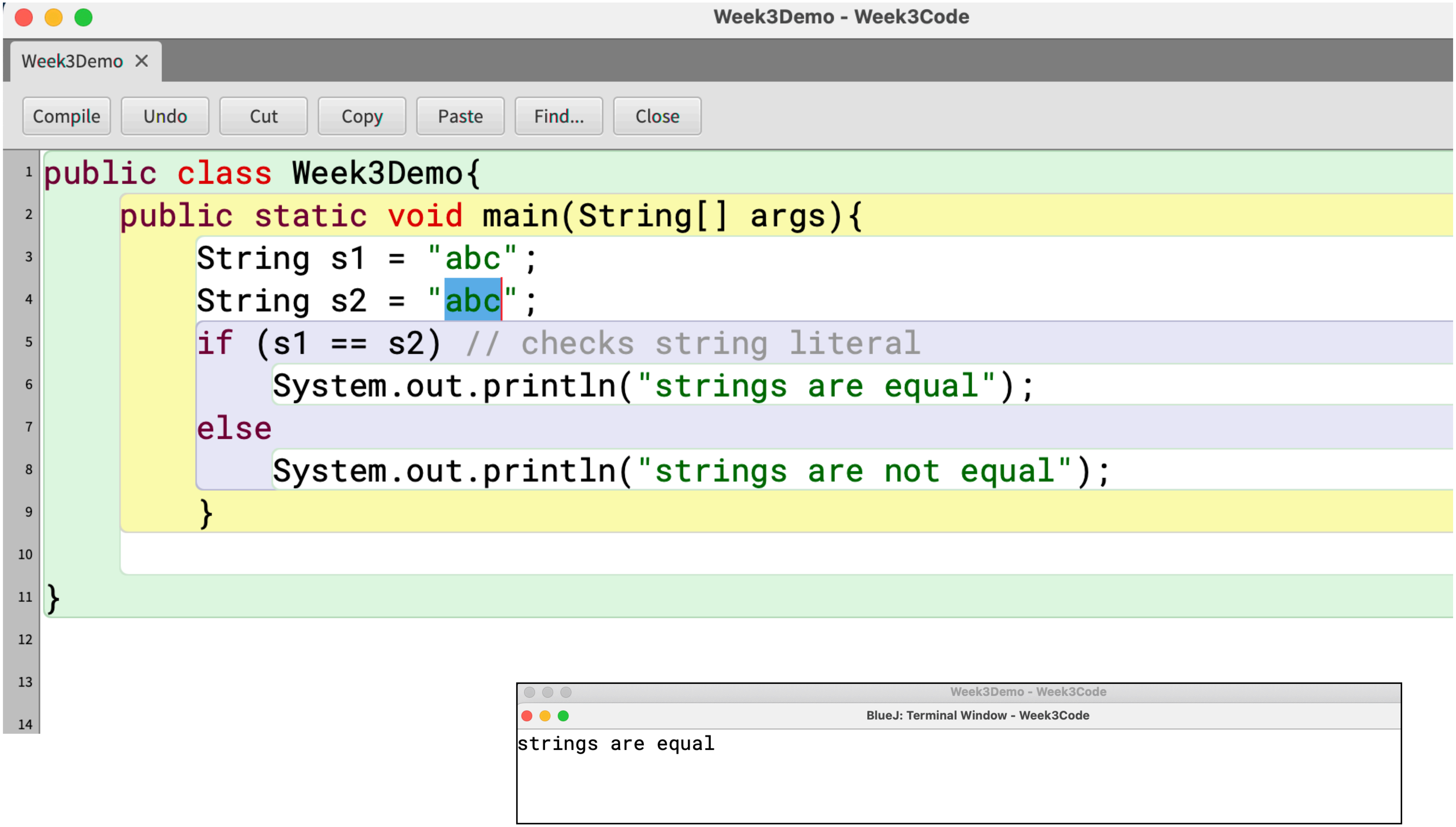


```java
public class Week3Demo{
    public static void main(String[] args){
        String s1 = "abc";
        String s2 = "def";
        if (s1 == s2) // checks string literal
            System.out.println("strings are equal");
        else
            System.out.println("strings are not equal");
        }

}
```



```
strings are not equal
```

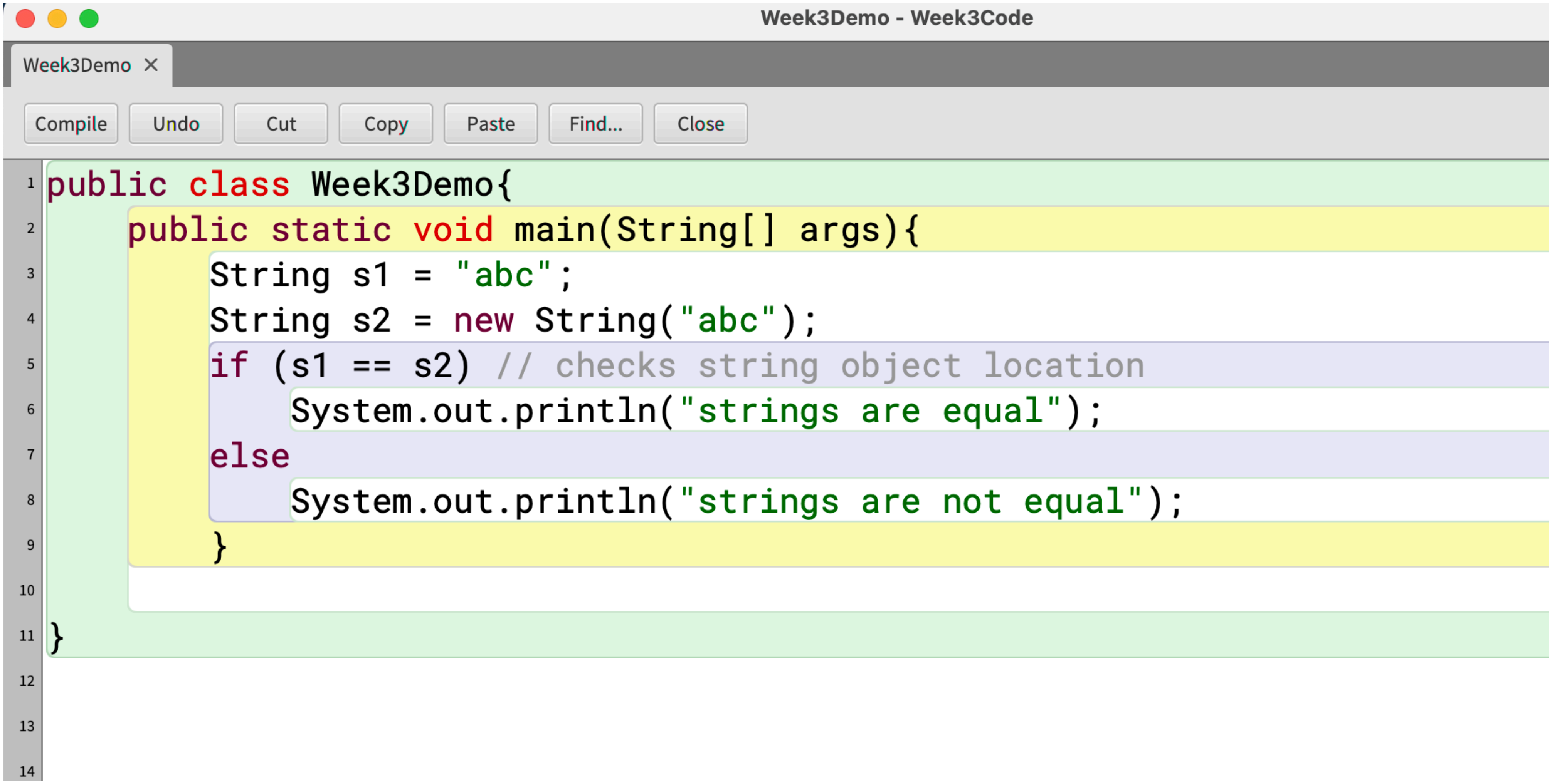
String Equality - Example 2



```java
public class Week3Demo{
    public static void main(String[] args){
        String s1 = "abc";
        String s2 = "abc";
        if (s1 == s2) // checks string literal
            System.out.println("strings are equal");
        else
            System.out.println("strings are not equal");
        }

}
```

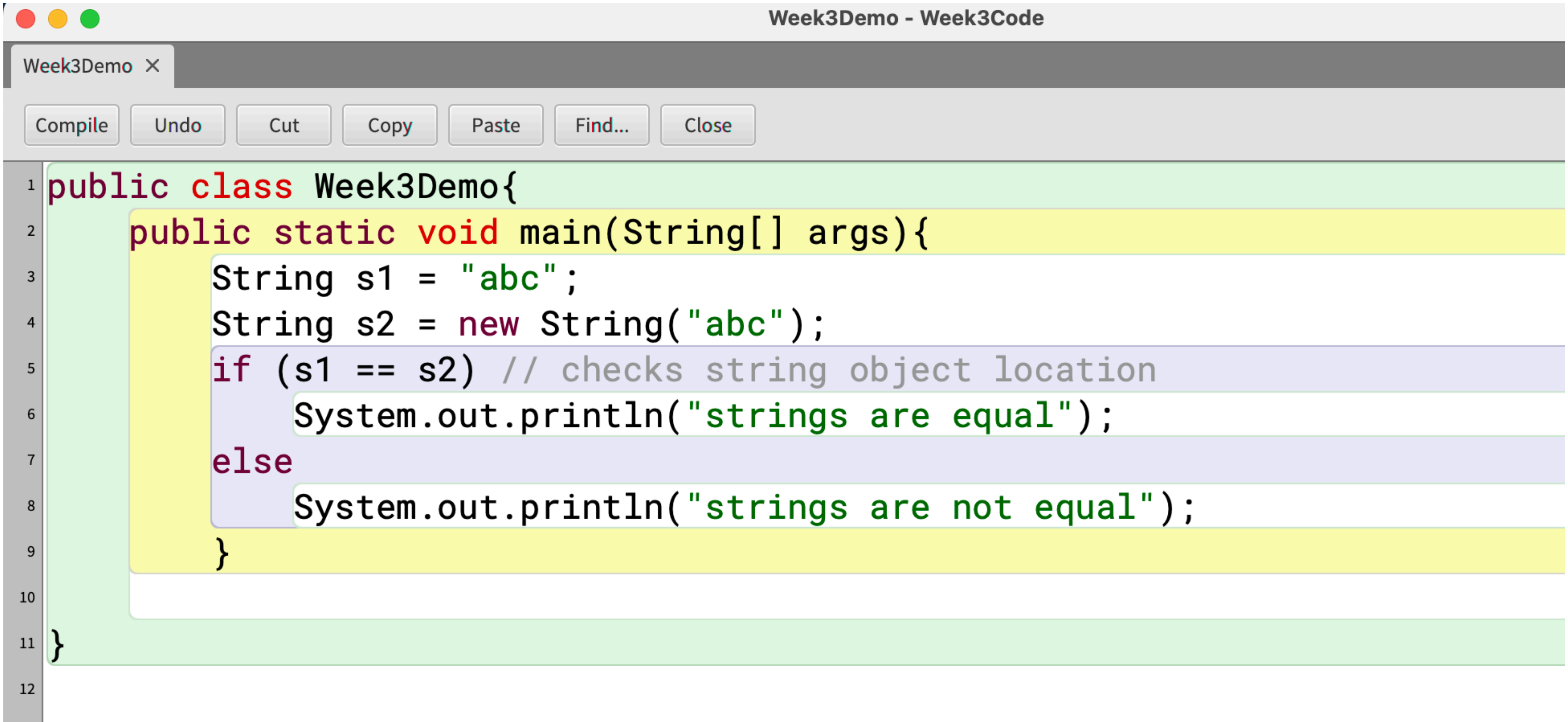
```
strings are equal
```

String Equality - Example 3

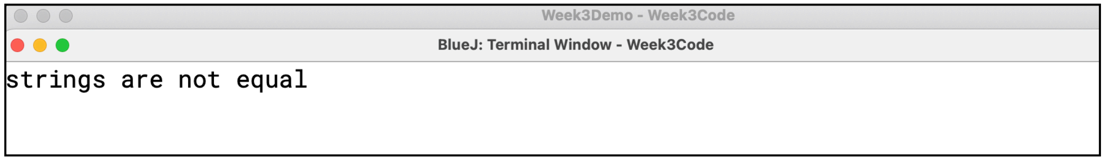


```java
public class Week3Demo{
    public static void main(String[] args){
        String s1 = "abc";
        String s2 = new String("abc");
        if (s1 == s2) // checks string object location
            System.out.println("strings are equal");
        else
            System.out.println("strings are not equal");
        }

}
```

String Equality - Example 3



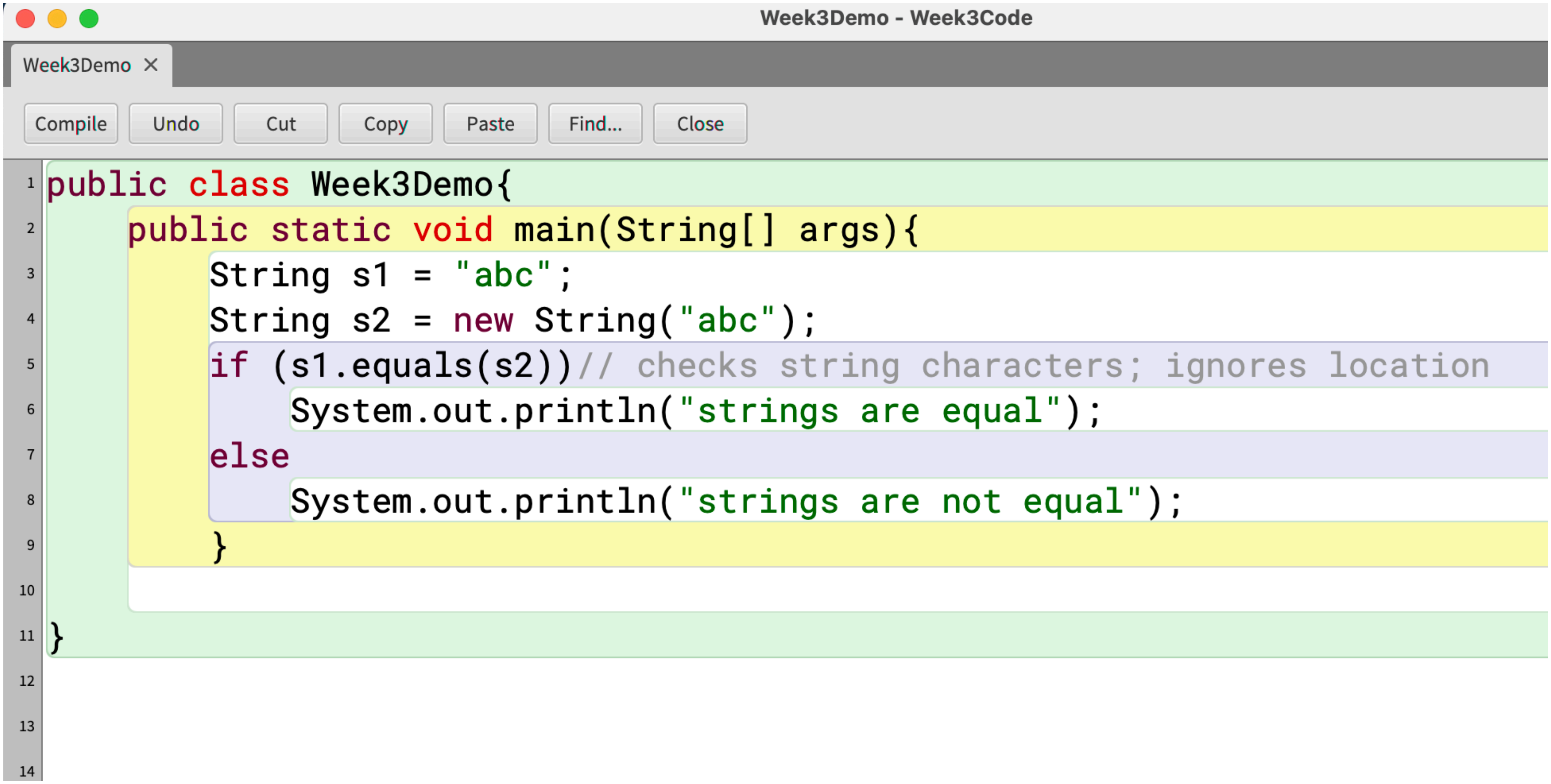
```java
public class Week3Demo{
    public static void main(String[] args){
        String s1 = "abc";
        String s2 = new String("abc");
        if (s1 == s2) // checks string object location
            System.out.println("strings are equal");
        else
            System.out.println("strings are not equal");
        }

}
```



```
strings are not equal
```

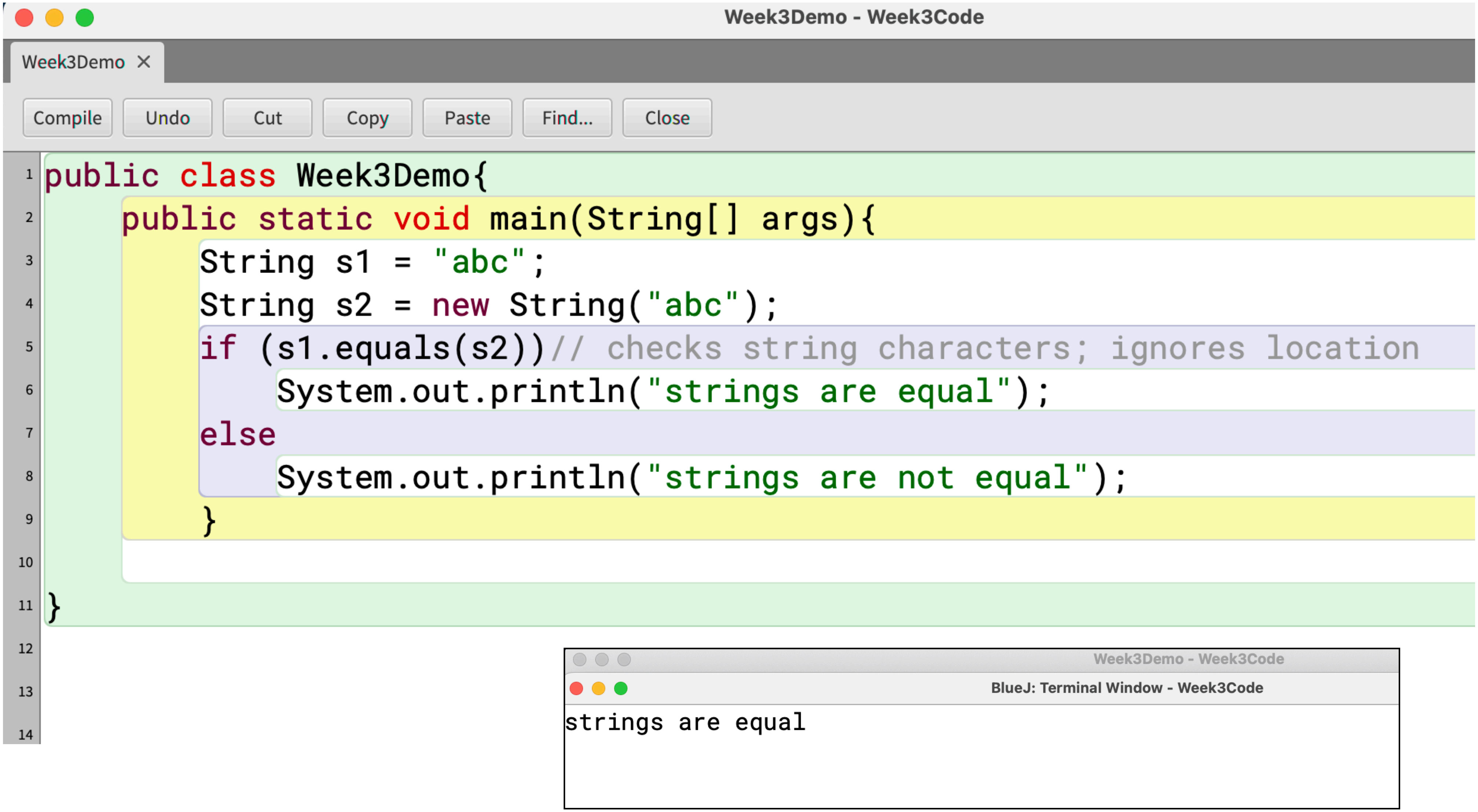
✗ But this output could be conceptually incorrect

String Equality - Example 4



```
public class Week3Demo{
    public static void main(String[] args){
        String s1 = "abc";
        String s2 = new String("abc");
        if (s1.equals(s2))// checks string characters; ignores location
            System.out.println("strings are equal");
        else
            System.out.println("strings are not equal");
        }

}
```

String Equality - Example 4



```
public class Week3Demo{
    public static void main(String[] args){
        String s1 = "abc";
        String s2 = new String("abc");
        if (s1.equals(s2))// checks string characters; ignores location
            System.out.println("strings are equal");
        else
            System.out.println("strings are not equal");
        }

}
```

```
strings are equal
```

Why? Because the String class provides its own equals method

Object Equality

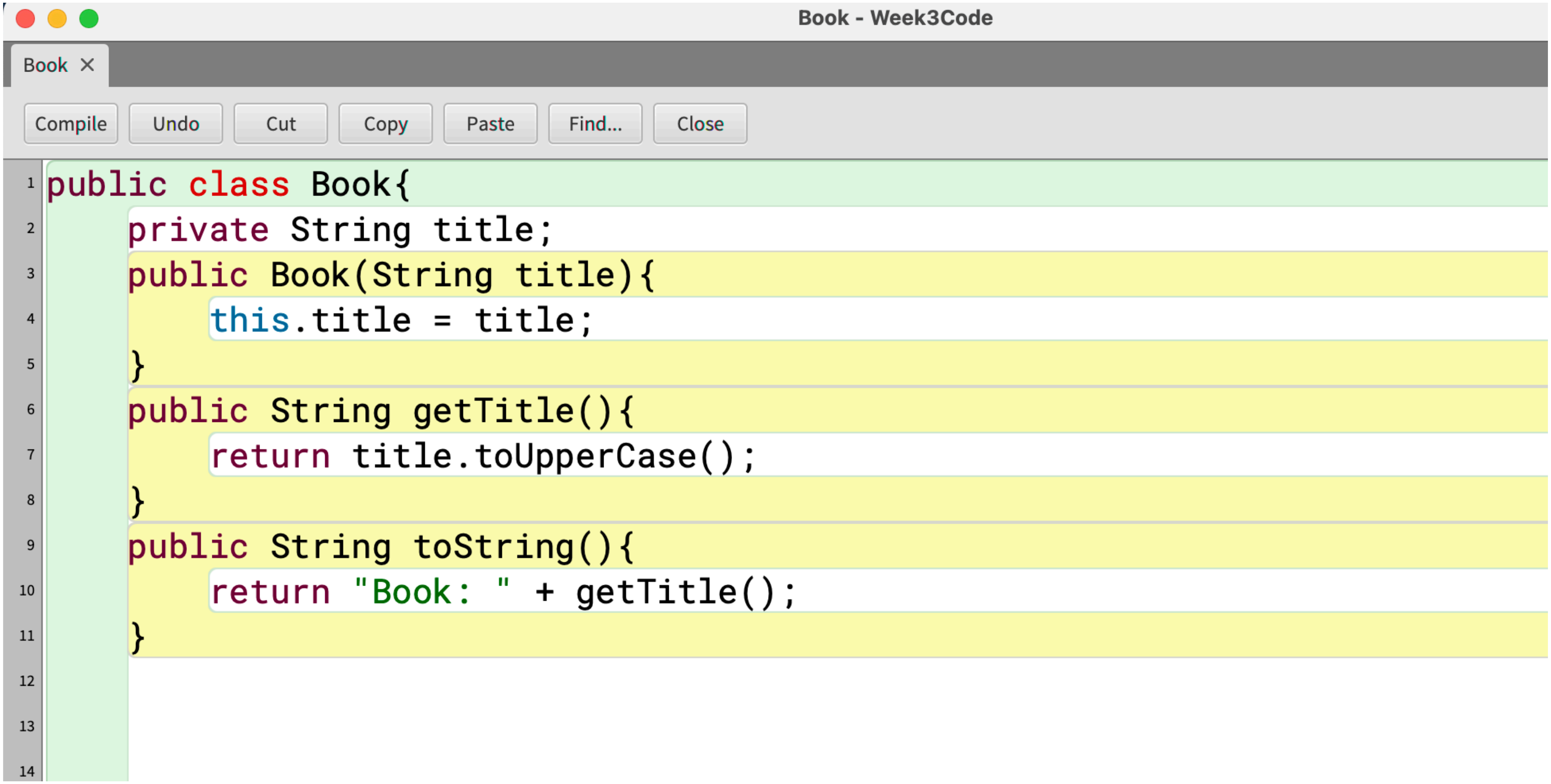
Consider the equals( ) method of the Object class:

```
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.

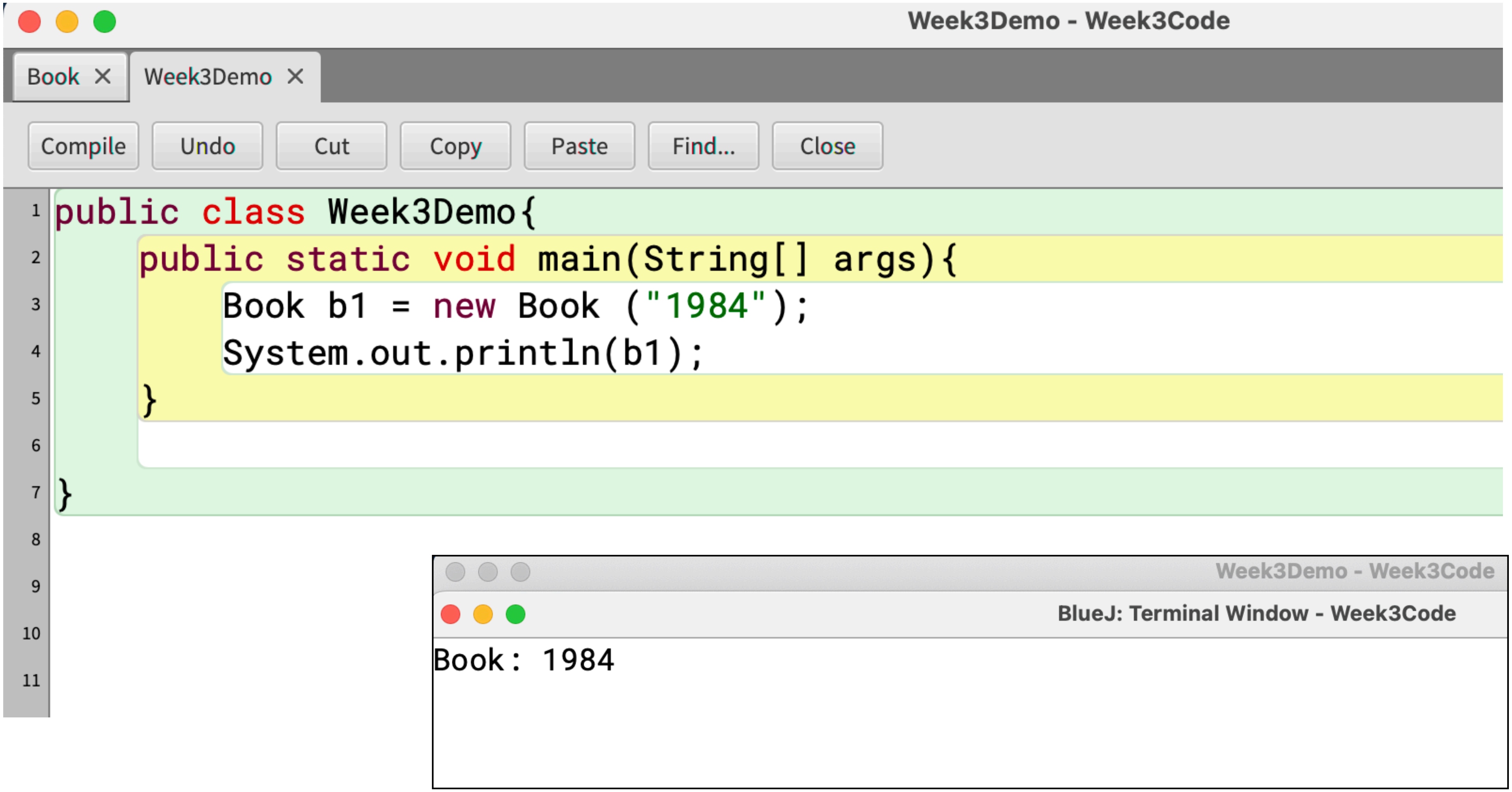
This method behaves just like = =

Book.java



```java
public class Book{
    private String title;
    public Book(String title){
        this.title = title;
    }
    public String getTitle(){
        return title.toUpperCase();
    }
    public String toString(){
        return "Book: " + getTitle();
    }
```

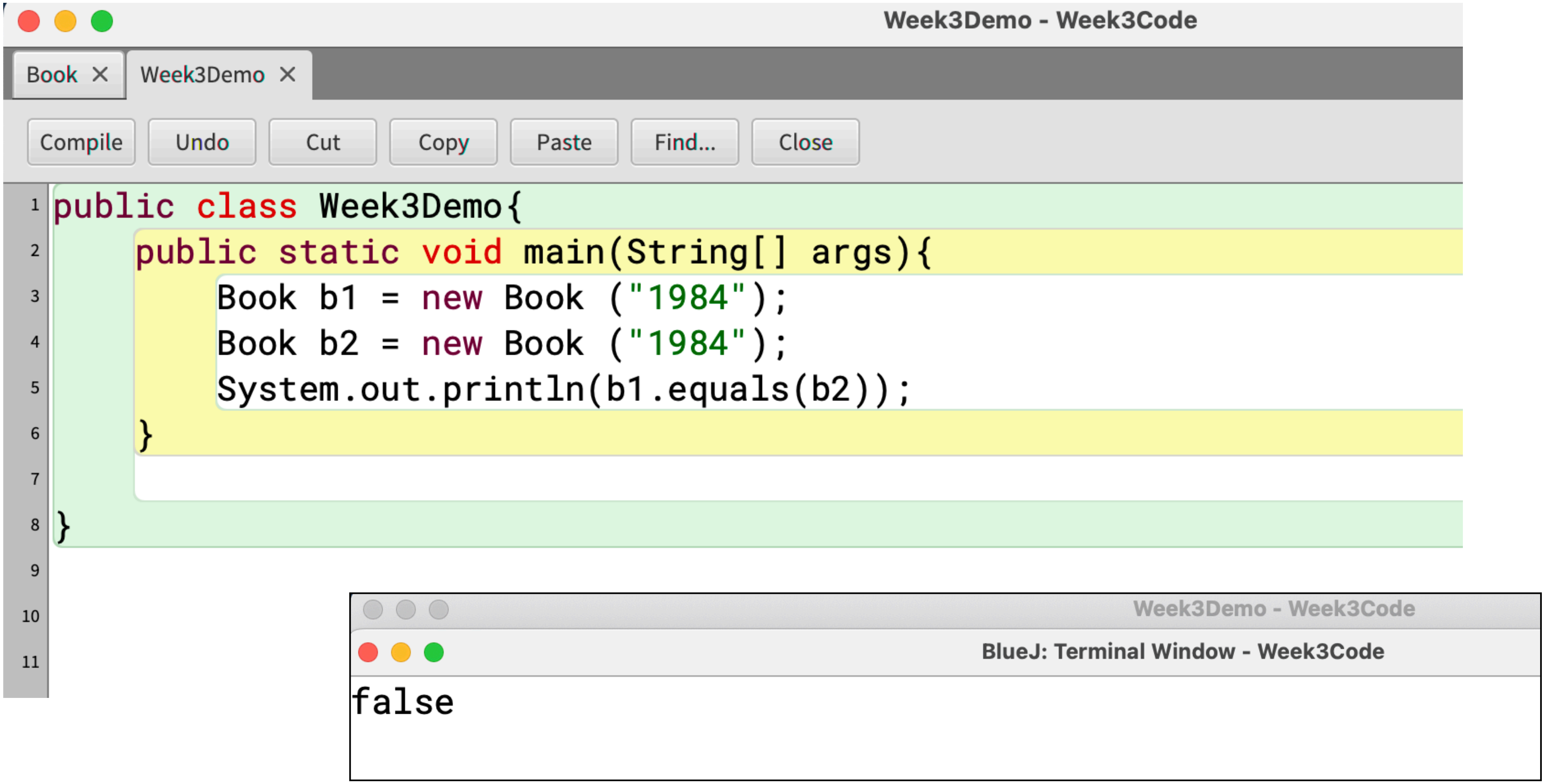
Runner - Book.java



```java
public class Week3Demo{
    public static void main(String[] args){
        Book b1 = new Book ("1984");
        System.out.println(b1);
    }

}
```

```
Book: 1984
```

Runner - Book.java

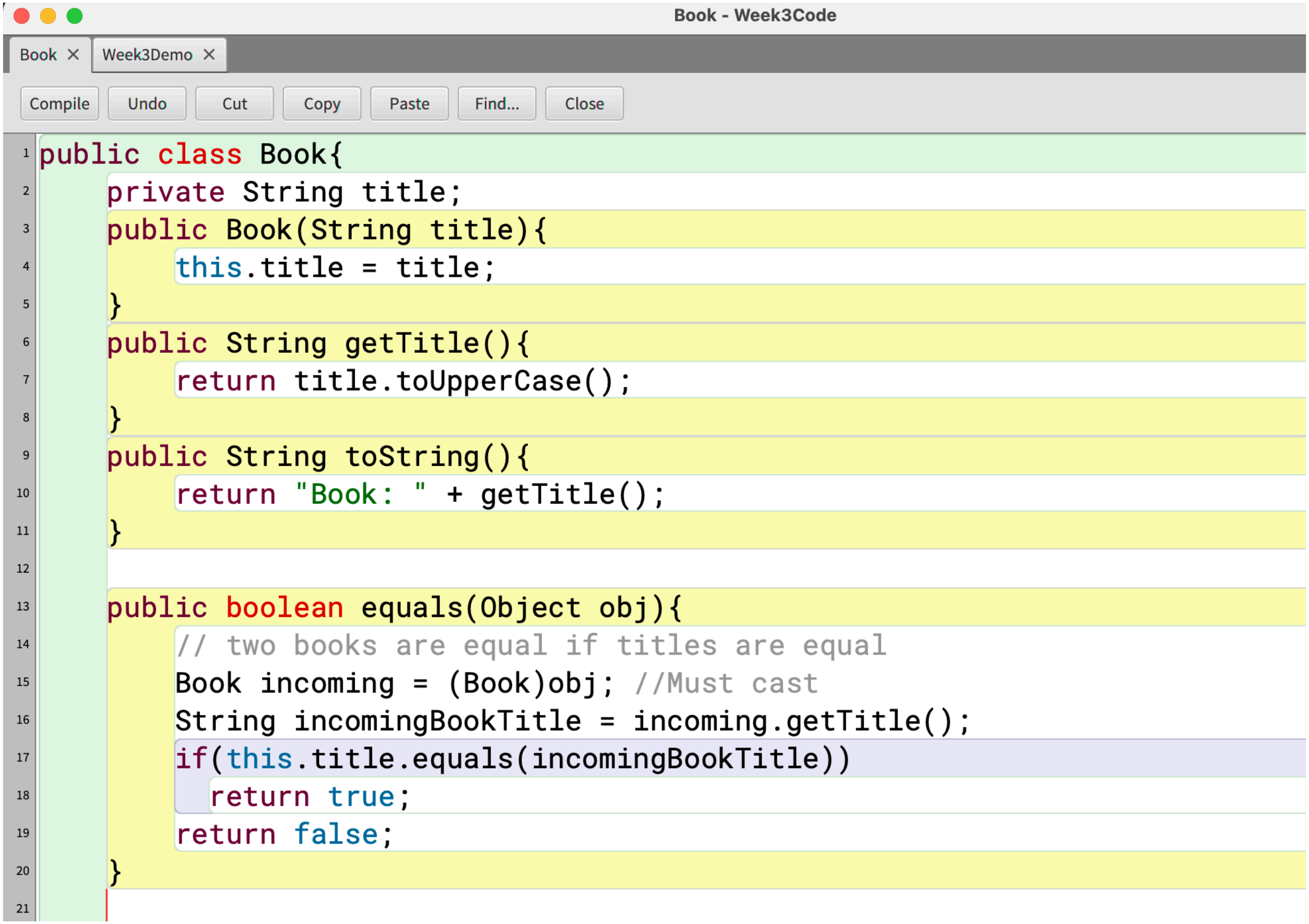


```java
public class Week3Demo{
    public static void main(String[] args){
        Book b1 = new Book ("1984");
        Book b2 = new Book ("1984");
        System.out.println(b1.equals(b2));
    }

}
```

```
false
```

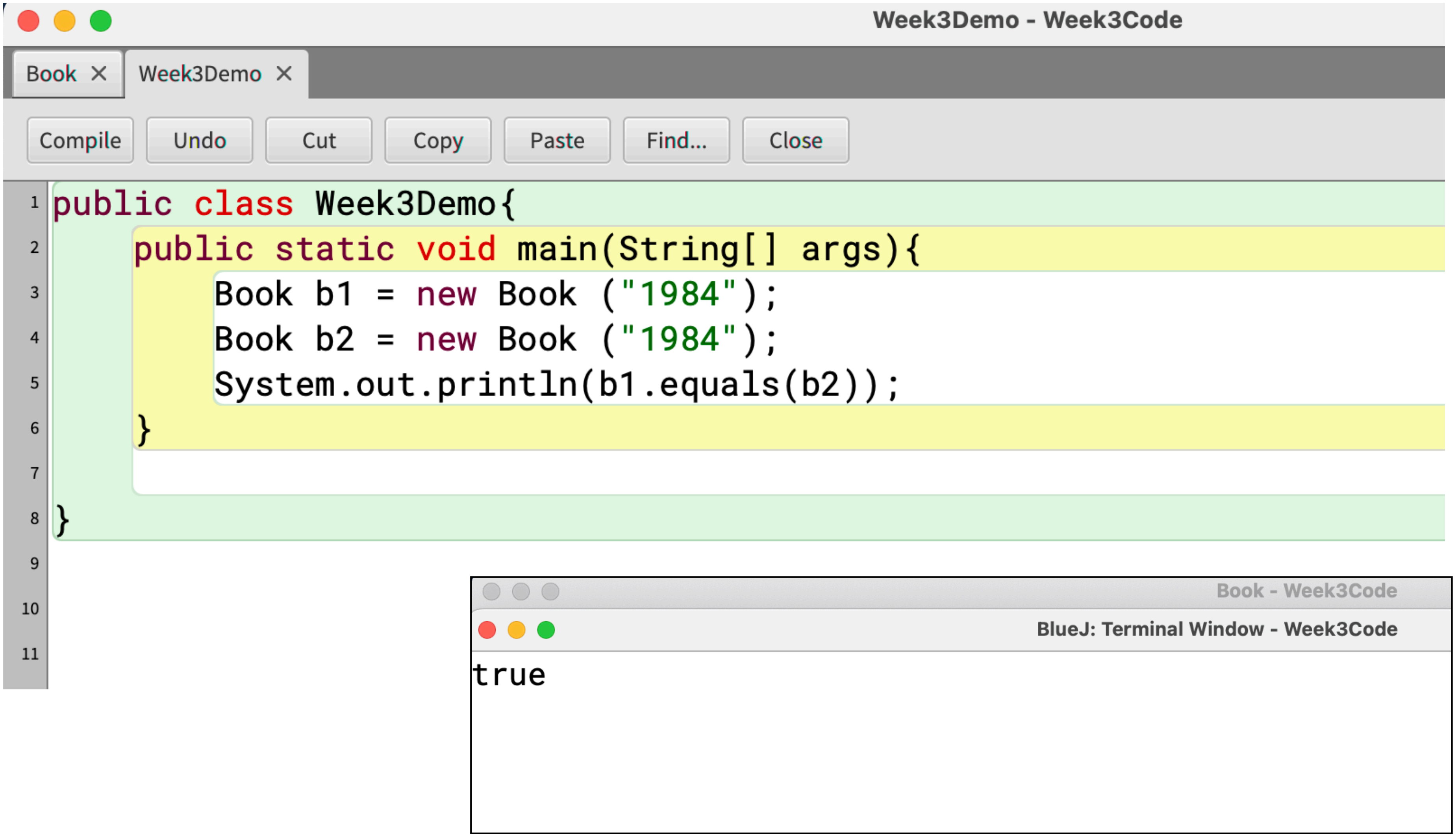
Why? Because the Object class provides an equals method.
It checks the memory location of the objects and compares those

Book.java - Custom equals( )



```
public class Book{
    private String title;
    public Book(String title){
        this.title = title;
    }
    public String getTitle(){
        return title.toUpperCase();
    }
    public String toString(){
        return "Book: " + getTitle();
    }

    public boolean equals(Object obj){
        // two books are equal if titles are equal
        Book incoming = (Book)obj; //Must cast
        String incomingBookTitle = incoming.getTitle();
        if(this.title.equals(incomingBookTitle))
            return true;
        return false;
    }

```

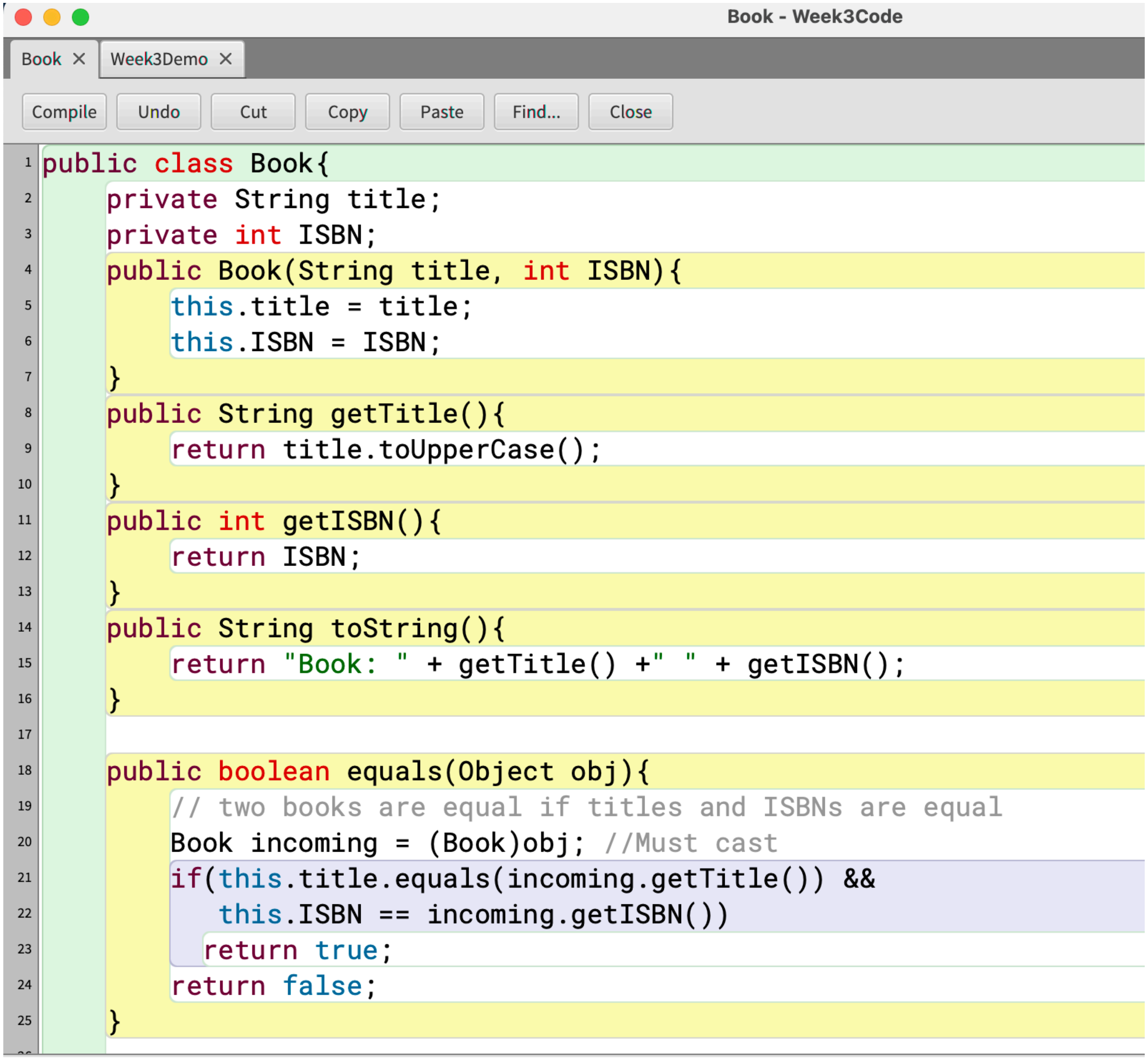
Book.java - Custom equals( ) - Runner



```java
public class Week3Demo{
    public static void main(String[] args){
        Book b1 = new Book ("1984");
        Book b2 = new Book ("1984");
        System.out.println(b1.equals(b2));
    }

}
```

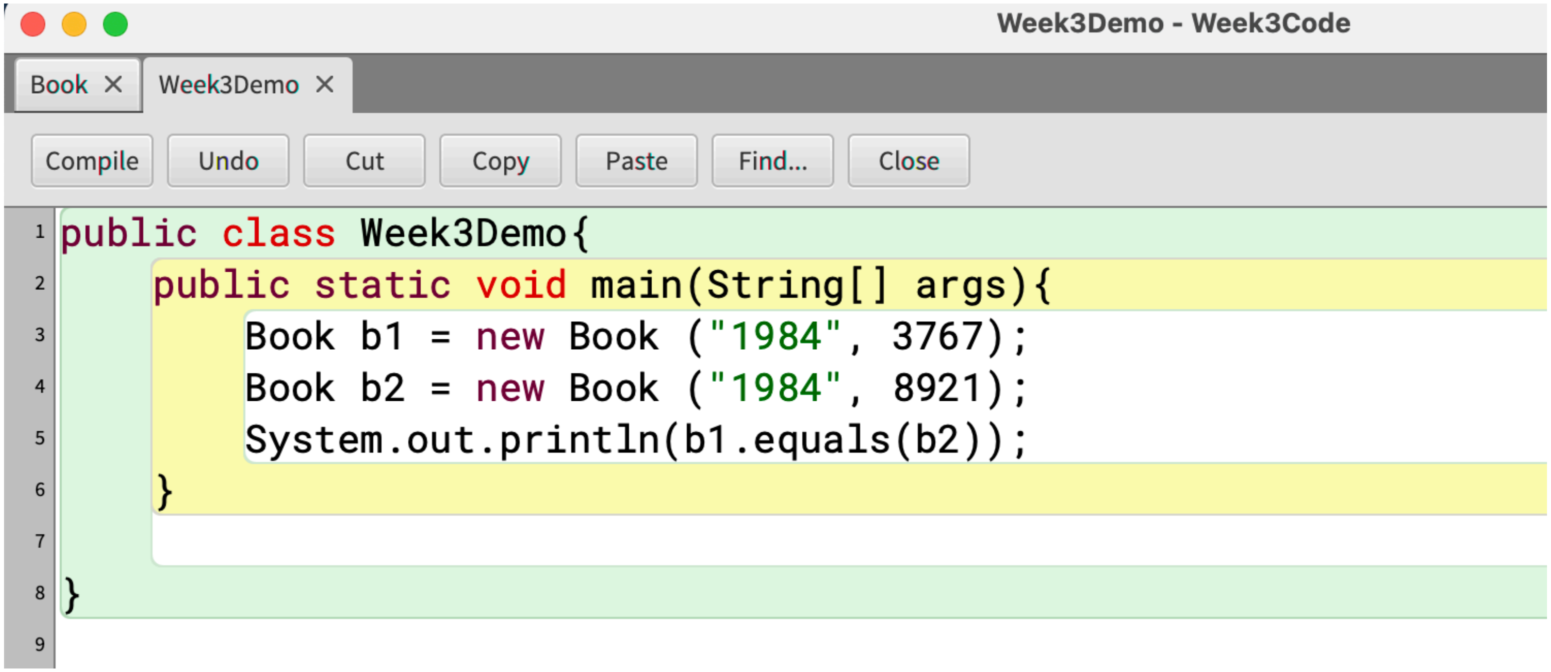
```
true
```

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

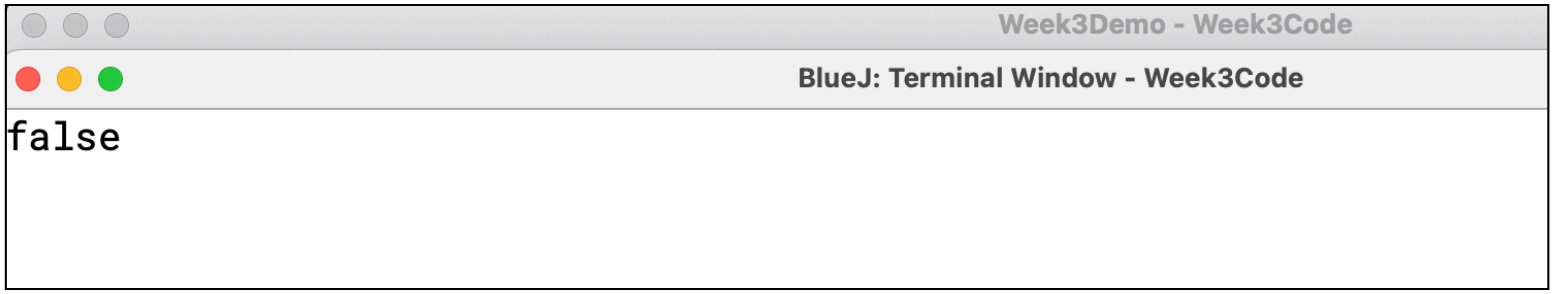


```java
public class Book{
    private String title;
    private int ISBN;
    public Book(String title, int ISBN){
        this.title = title;
        this.ISBN = ISBN;
    }
    public String getTitle(){
        return title.toUpperCase();
    }
    public int getISBN(){
        return ISBN;
    }
    public String toString(){
        return "Book: " + getTitle() +" " + getISBN();
    }

    public boolean equals(Object obj){
        // two books are equal if titles and ISBNs are equal
        Book incoming = (Book)obj; //Must cast
        if(this.title.equals(incoming.getTitle()) &&
           this.ISBN == incoming.getISBN())
          return true;
        return false;
    }
```

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



```java
public class Week3Demo{
    public static void main(String[] args){
        Book b1 = new Book ("1984", 3767);
        Book b2 = new Book ("1984", 8921);
        System.out.println(b1.equals(b2));
    }

}
```



```
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

Reading: Page 105, Chapter 3 - G. Booch; Page 131

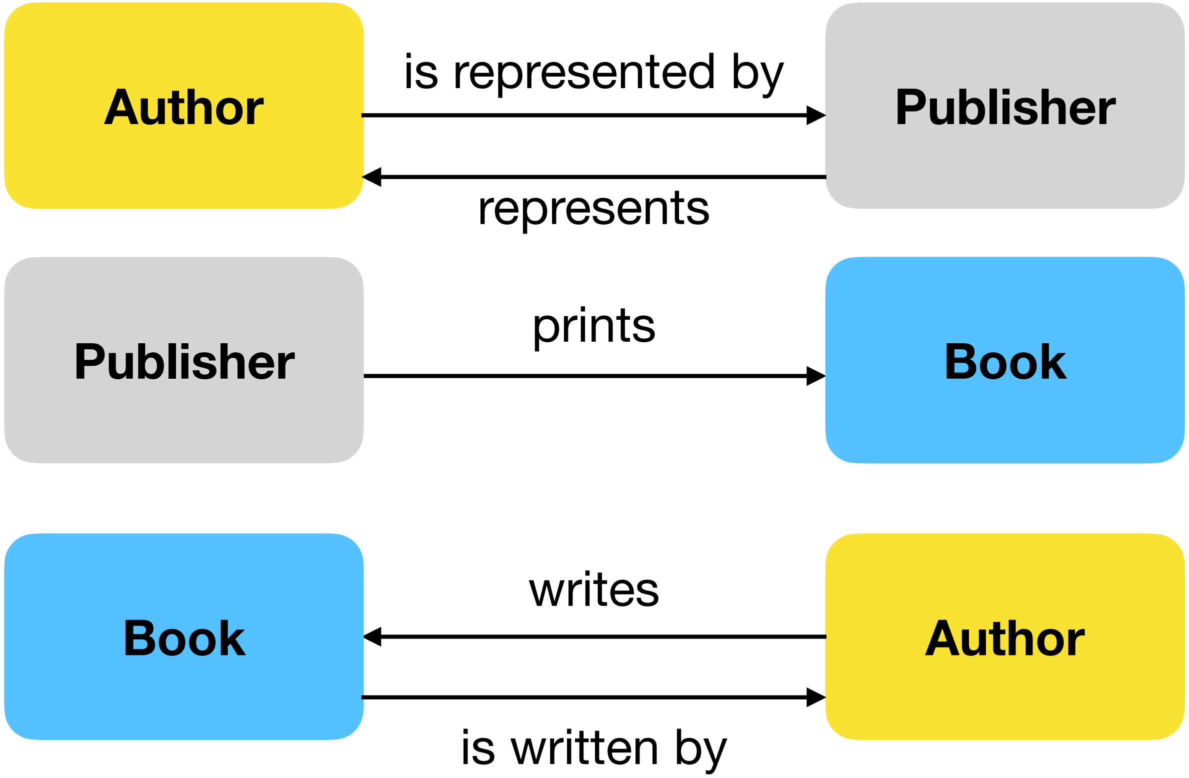
Kinds of Relationships

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.



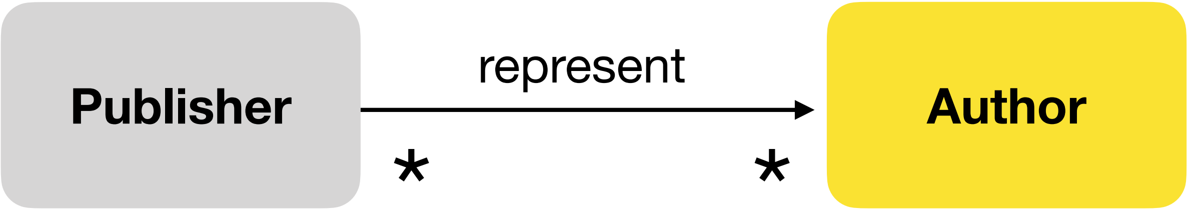
Cardinality

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

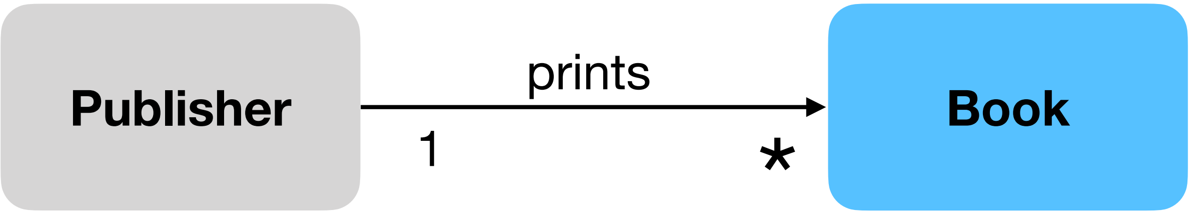
Three common types of cardinality are:

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

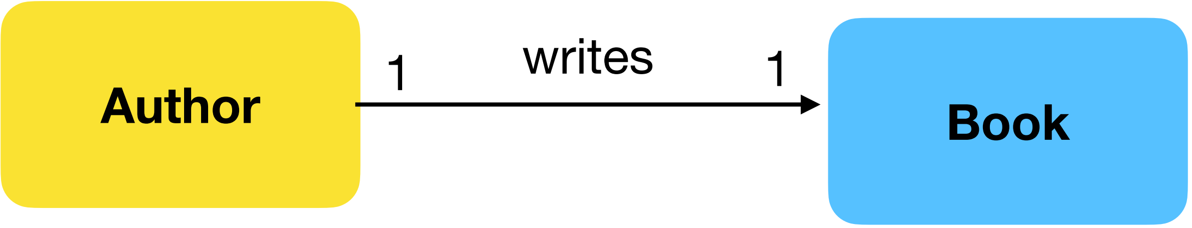
Example- Associations + Cardinalities



many-to-many association

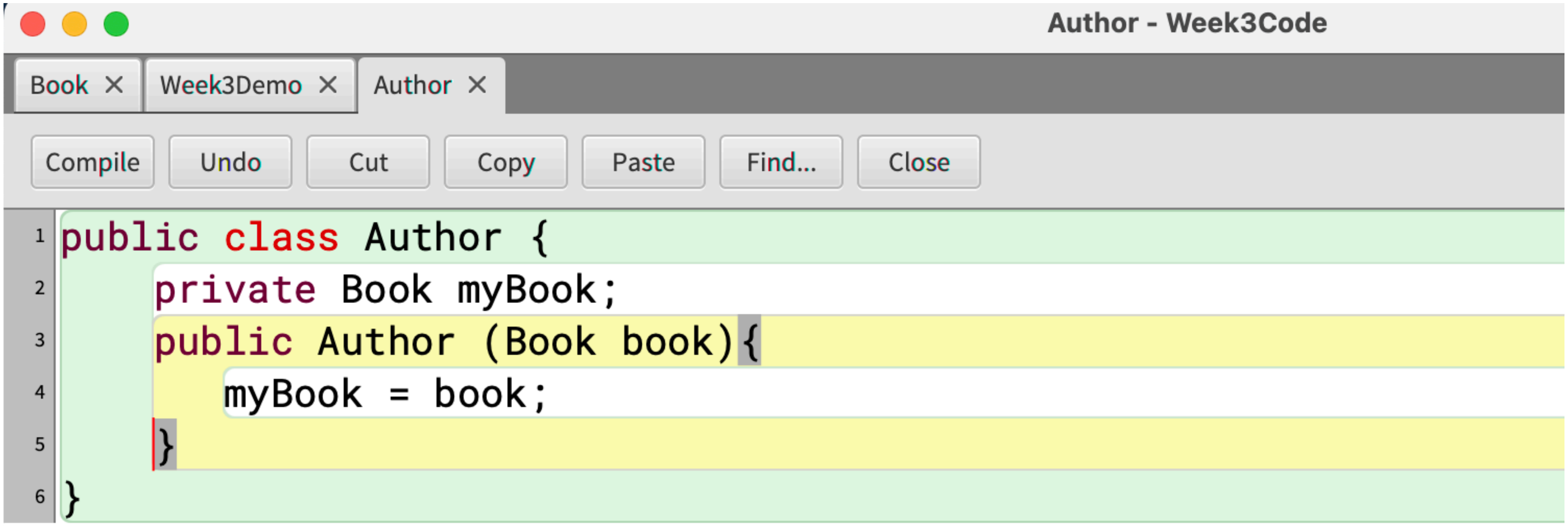


one-to-many association

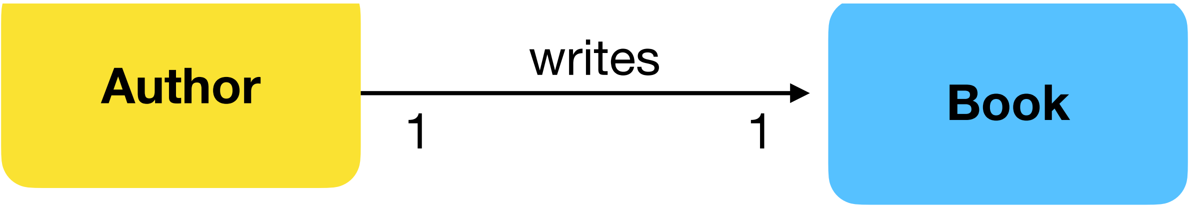


one-to-one association

Example - Implementing Associations 1:1



```java
public class Author {
        private Book myBook;
        public Author (Book book){
            myBook = book;
        }
}
```


Author

writes

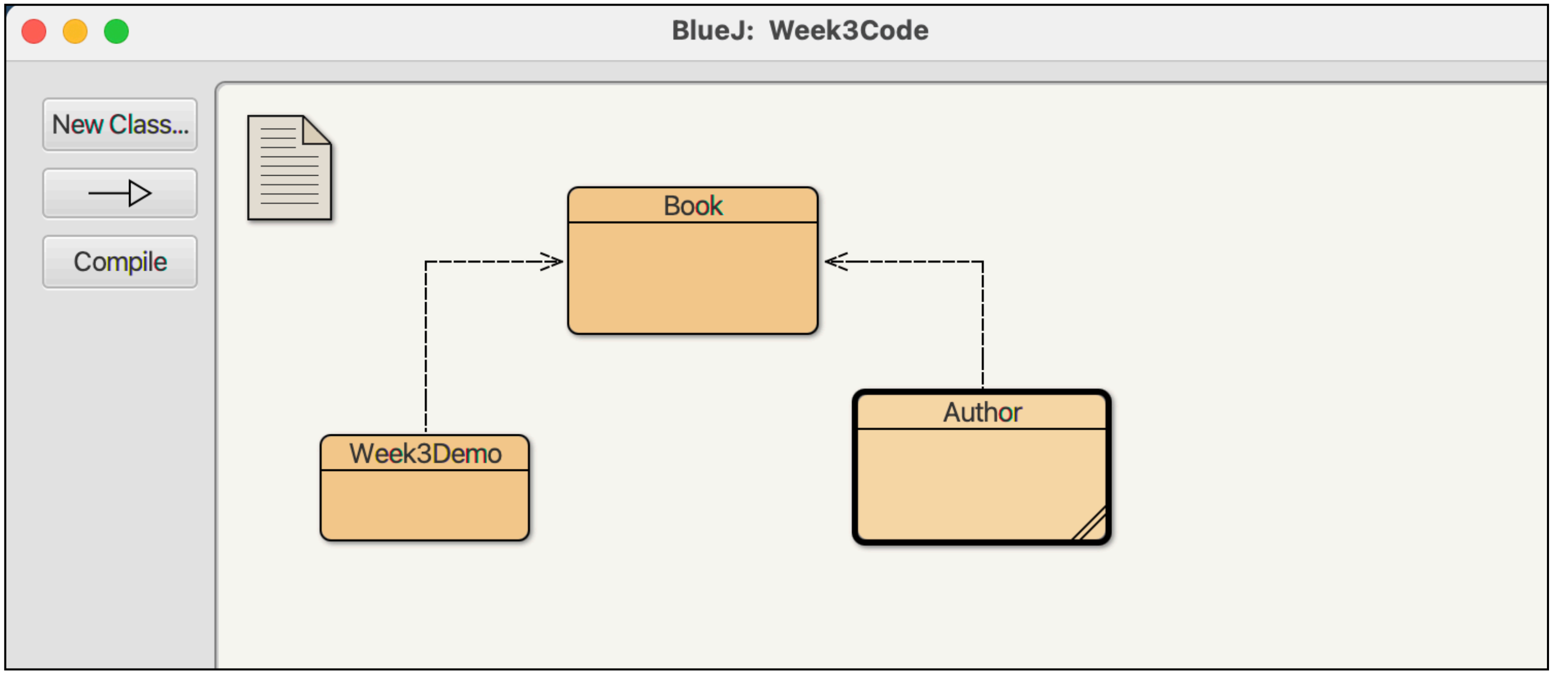
1

1

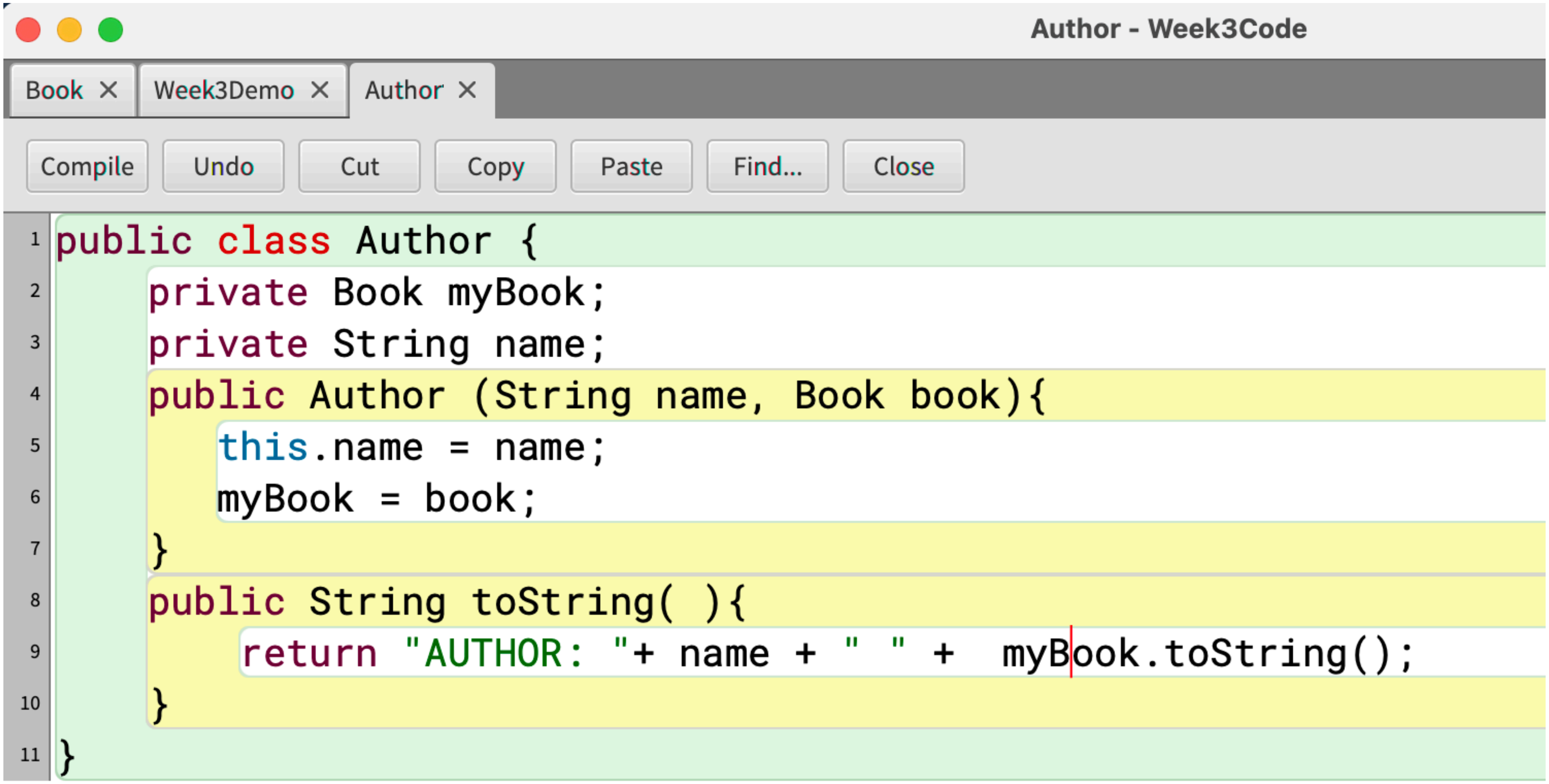
Book

one-to-one association

Example - Implementing Associations 1:1

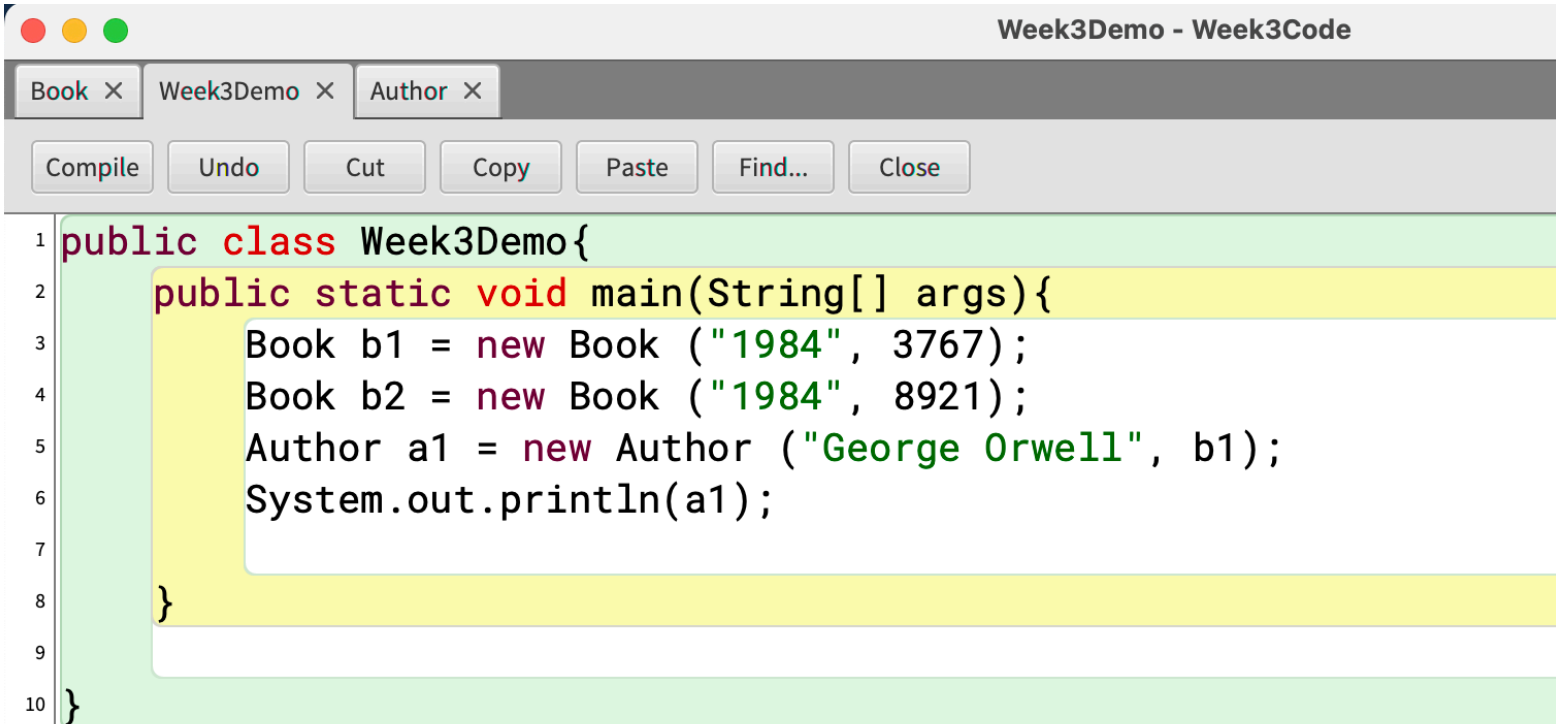


Example - Implementing Associations 1:1

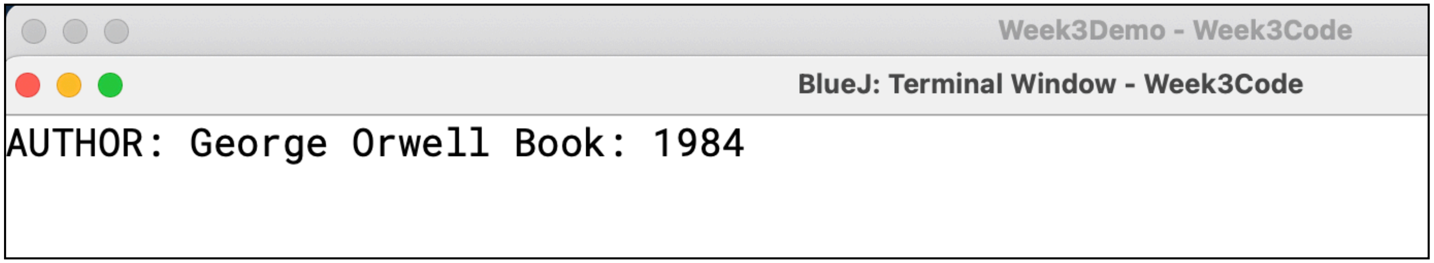


```java
public class Author {
    private Book myBook;
    private String name;
    public Author (String name, Book book){
        this.name = name;
        myBook = book;
    }
    public String toString( ){
         return "AUTHOR: "+ name + " " +  myBook.toString();
    }
}
```

Example - Implementing Associations 1:1



```java
public class Week3Demo{
    public static void main(String[] args){
        Book b1 = new Book ("1984", 3767);
        Book b2 = new Book ("1984", 8921);
        Author a1 = new Author ("George Orwell", b1);
        System.out.println(a1);

    }

}
```



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

Question

1. Indicate if the statement is TRUE/FALSE

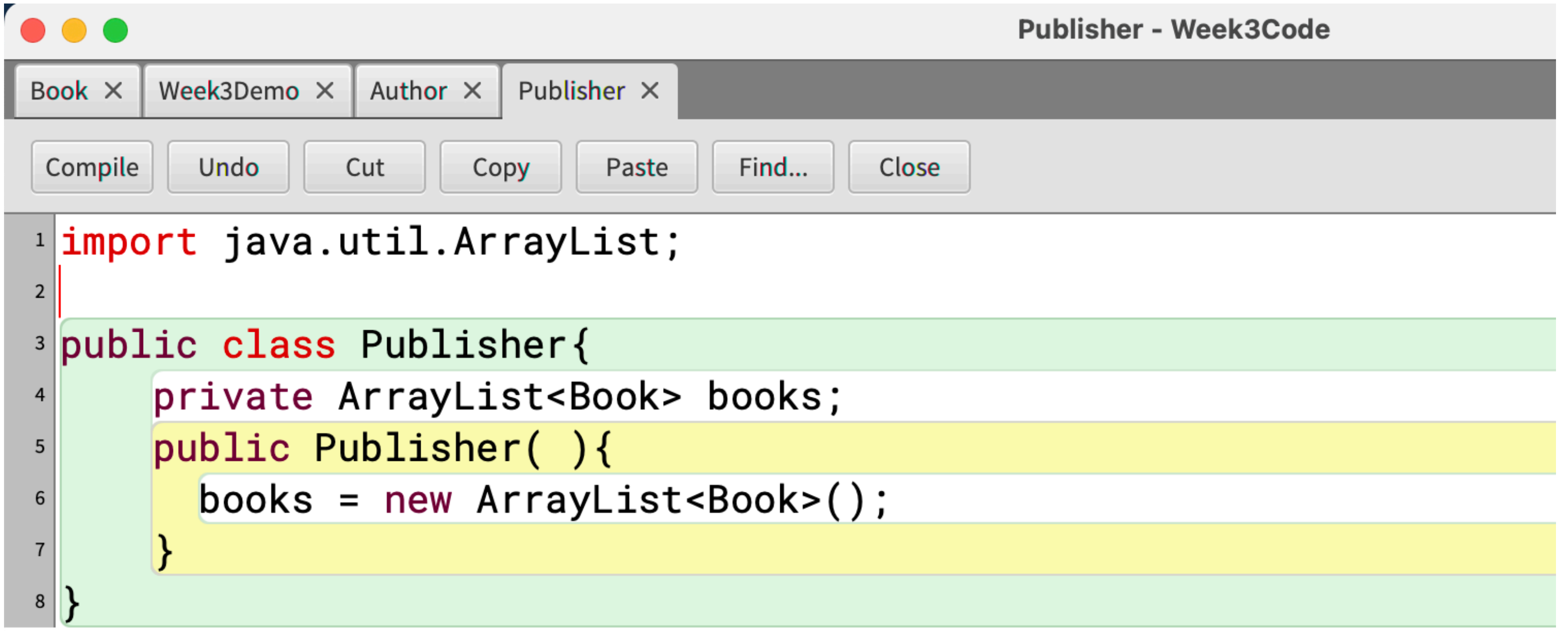
(a) One to one relationships are always bidirectional

(b) One to one relationships are implemented with a collection

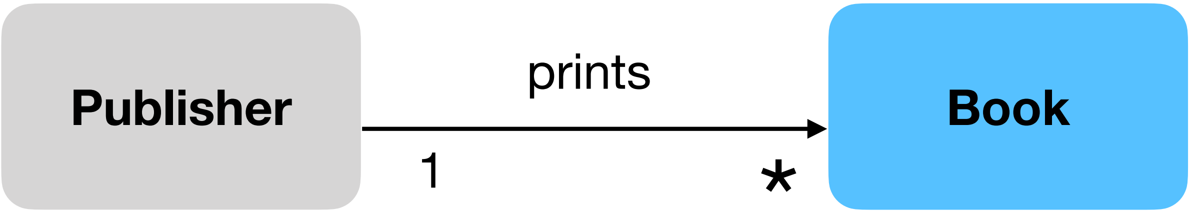
(c) One to one relationships are created with object variables

(d) One to one relationships are dependencies between classes

Example - Implementing Associations 1: *

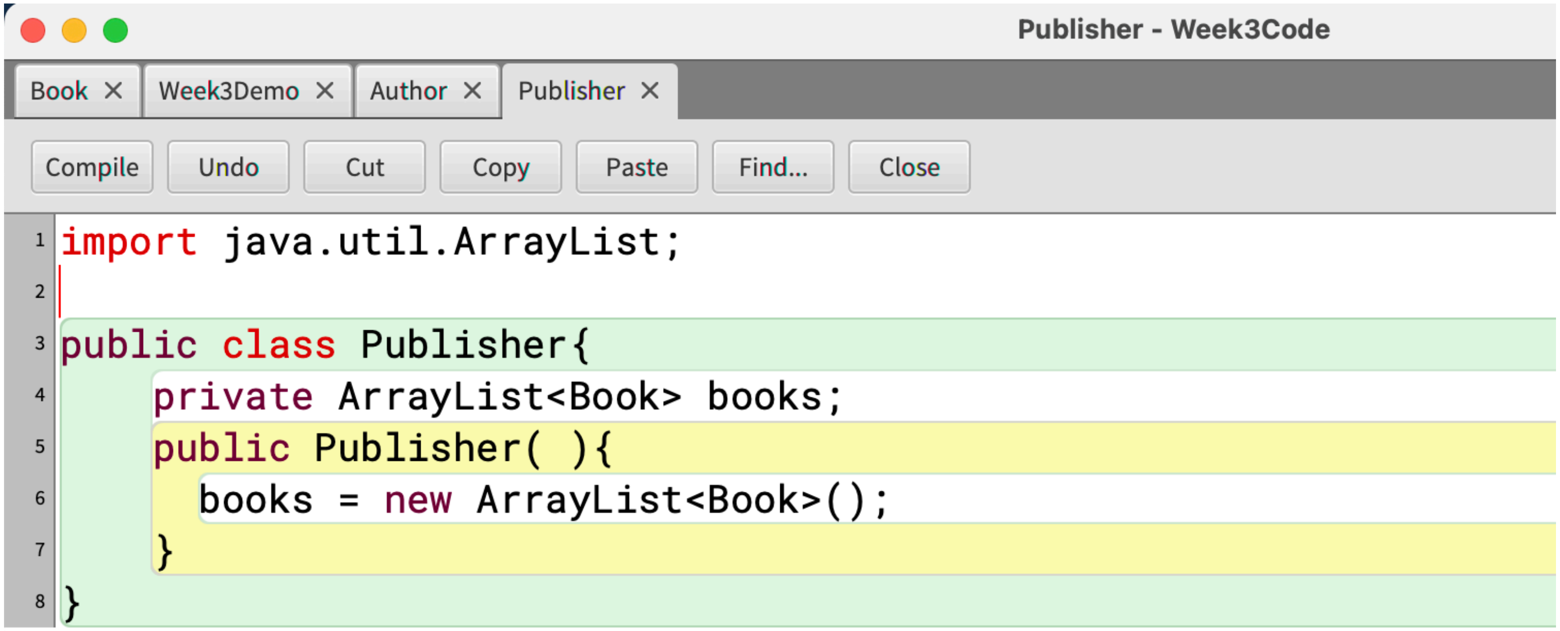


```java
import java.util.ArrayList;

public class Publisher{
    private ArrayList<Book> books;
    public Publisher( ){
      books = new ArrayList<Book>();
    }
}
```

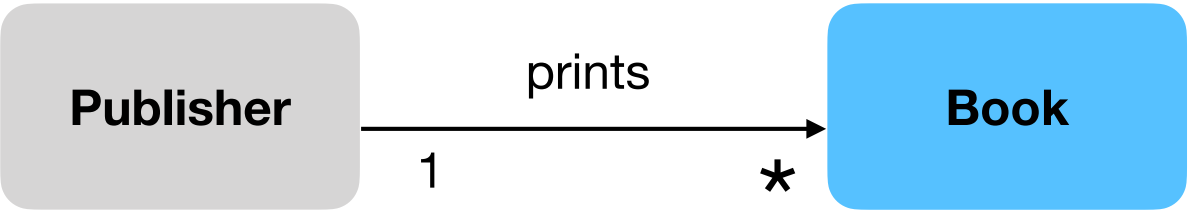


one-to-many association

Example - Implementing Associations 1: *

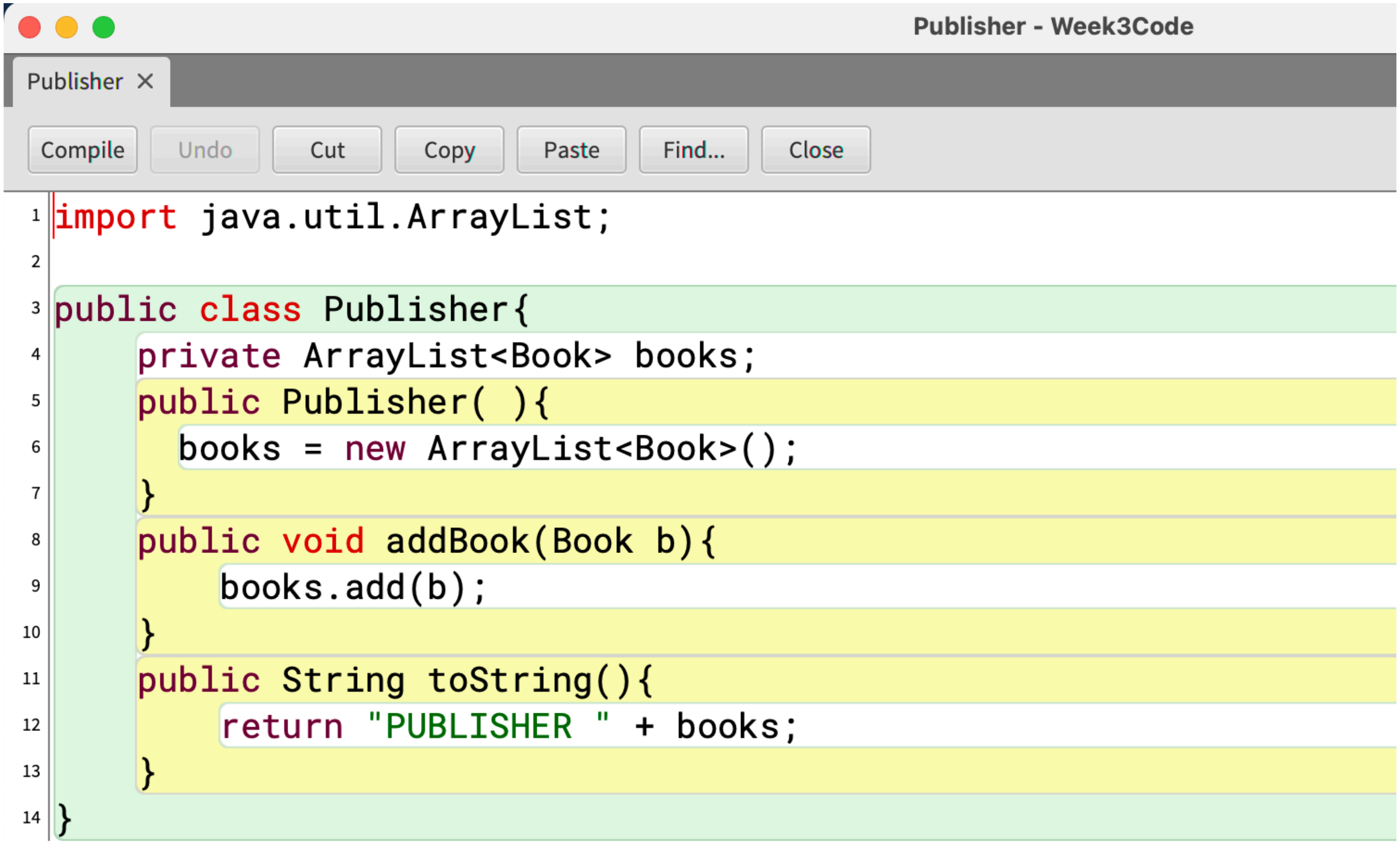


```java
import java.util.ArrayList;

public class Publisher{
    private ArrayList<Book> books;
    public Publisher( ){
      books = new ArrayList<Book>();
    }
}
```



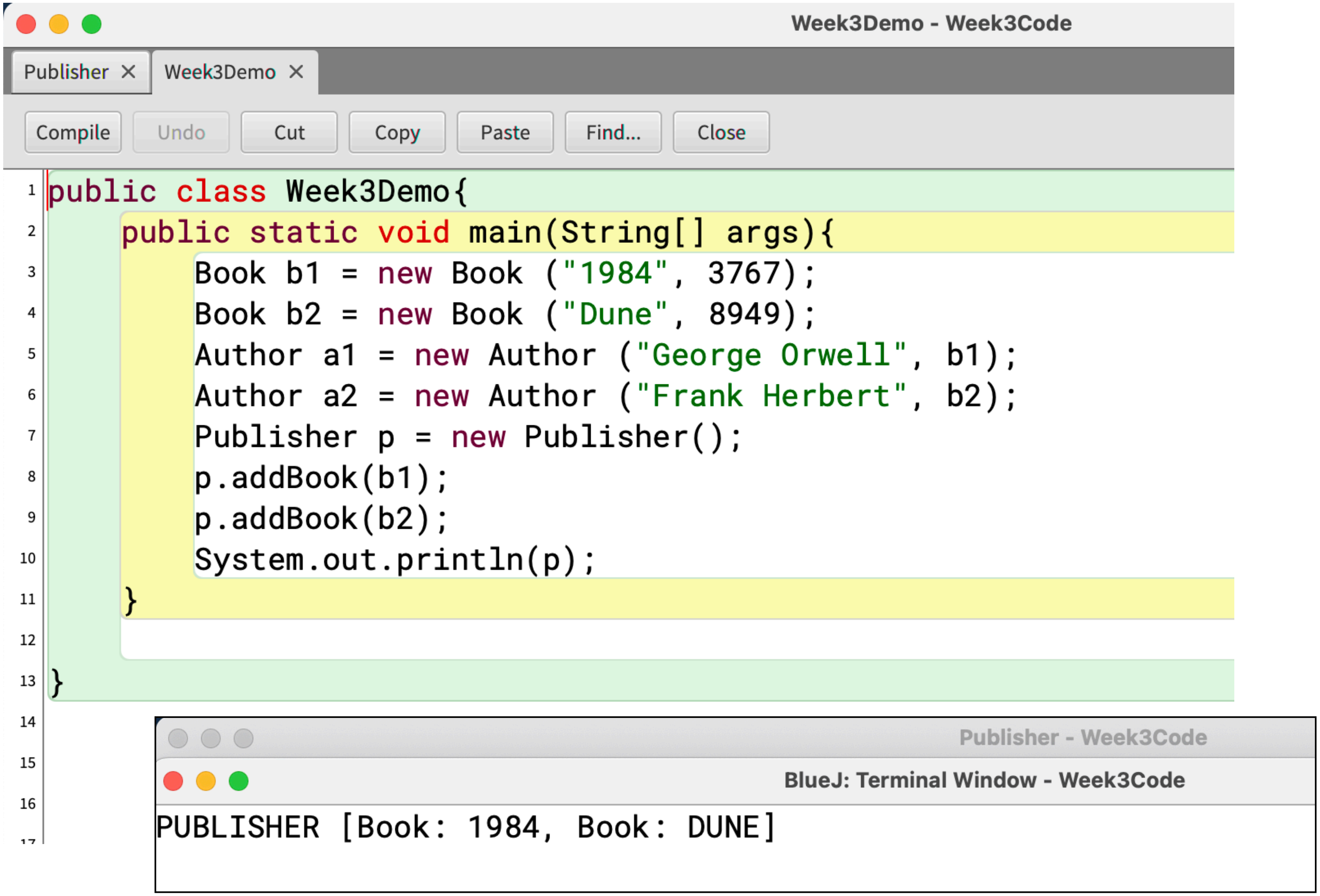
one-to-many association

Example - Implementing Associations 1: *



```
import java.util.ArrayList;

public class Publisher{
	private ArrayList<Book> books;
	public Publisher( ){
	  books = new ArrayList<Book>();
	}
	public void addBook(Book b){
		books.add(b);
	}
	public String toString(){
		return "PUBLISHER " + books;
	}
}
```

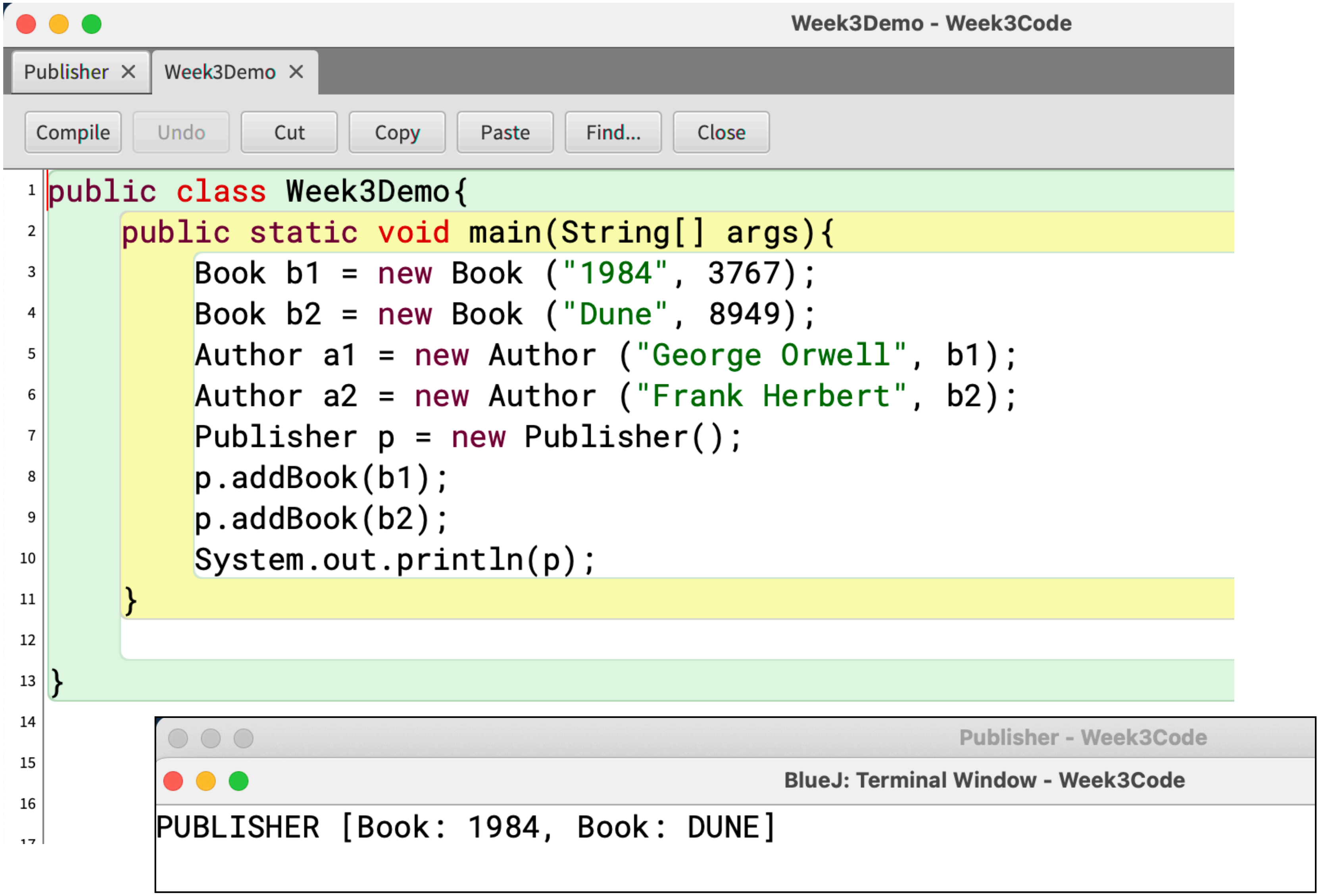
Example - Implementing Associations 1: *



```java
public class Week3Demo{
    public static void main(String[] args){
        Book b1 = new Book ("1984", 3767);
        Book b2 = new Book ("Dune", 8949);
        Author a1 = new Author ("George Orwell", b1);
        Author a2 = new Author ("Frank Herbert", b2);
        Publisher p = new Publisher();
        p.addBook(b1);
        p.addBook(b2);
        System.out.println(p);
    }

}
```

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

Example - Implementing Associations 1: *

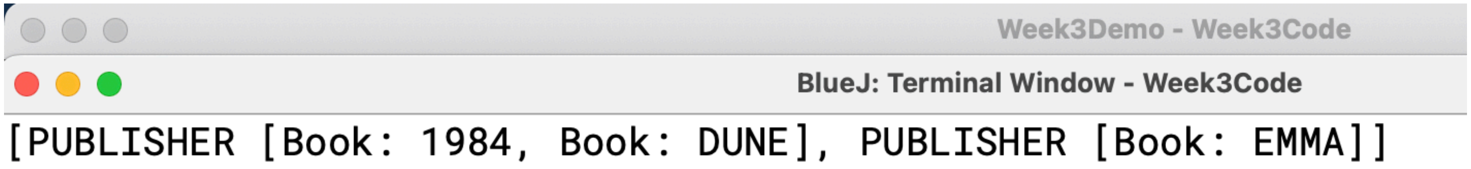


```java
public class Week3Demo{
    public static void main(String[] args){
        Book b1 = new Book ("1984", 3767);
        Book b2 = new Book ("Dune", 8949);
        Author a1 = new Author ("George Orwell", b1);
        Author a2 = new Author ("Frank Herbert", b2);
        Publisher p = new Publisher();
        p.addBook(b1);
        p.addBook(b2);
        System.out.println(p);
    }

}
```

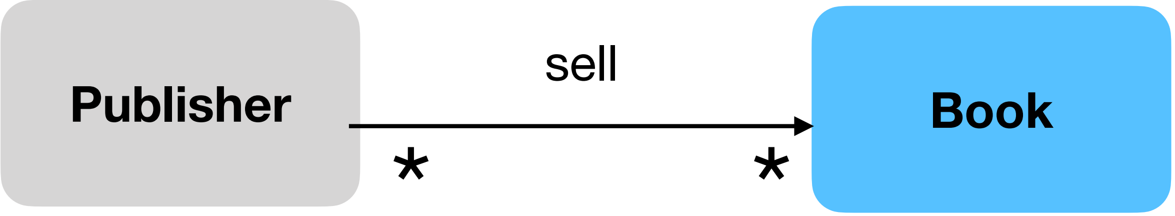
```
PUBLISHER [Book: 1984, Book: DUNE]
```

Example - Implementing Associations

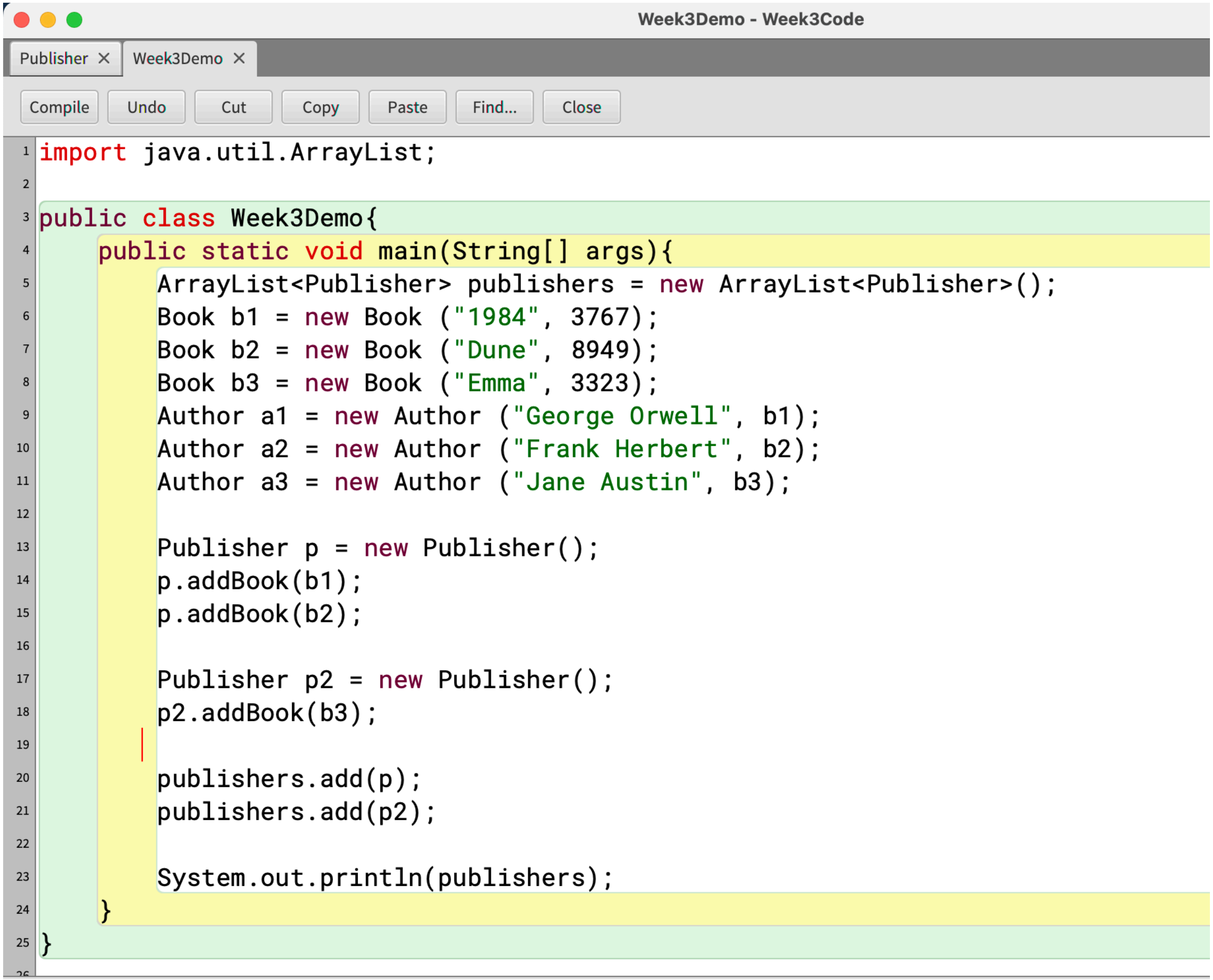
:



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



Example - Implementing Associations *:*



```java
import java.util.ArrayList;

public class Week3Demo{
    public static void main(String[] args){
        ArrayList<Publisher> publishers = new ArrayList<Publisher>();
        Book b1 = new Book ("1984", 3767);
        Book b2 = new Book ("Dune", 8949);
        Book b3 = new Book ("Emma", 3323);
        Author a1 = new Author ("George Orwell", b1);
        Author a2 = new Author ("Frank Herbert", b2);
        Author a3 = new Author ("Jane Austin", b3);

        Publisher p = new Publisher();
        p.addBook(b1);
        p.addBook(b2);

        Publisher p2 = new Publisher();
        p2.addBook(b3);

        publishers.add(p);
        publishers.add(p2);

        System.out.println(publishers);
    }
}
```

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