MIS 768: Advanced Software Concepts

Spring 2024

Classes (3)

Purpose

- Use objects in an application
- Practice the creation and usage of Enum
- Use objects in another model class for class collaboration
- Use a list of objects in a model class for class aggregation

1. Preparation

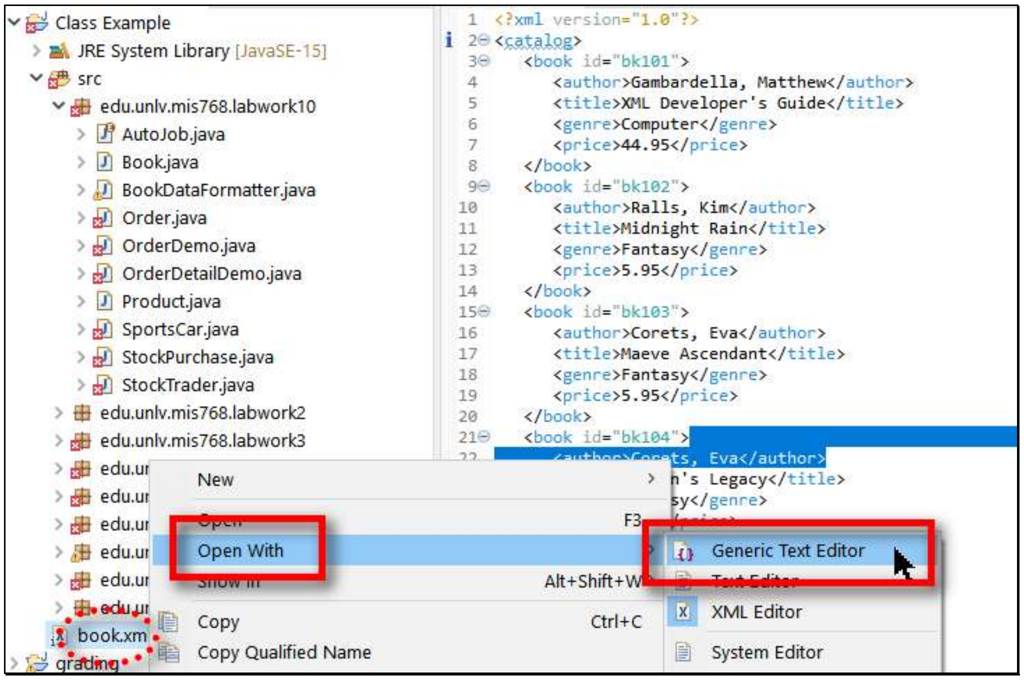
(1) Launch Eclipse, and set the workspace to your personal directory.

(2) Create a **package** to hold our source file. Select the folder **src** from the package navigator. Right click on the folder, and then select **New \ Package** from the popup menu.
Name the package as **edu.unlv.mis768.labwork10.**

(3) Download **10_lab_files.zip** from WebCampus. Extract the zip file and then import the .java files into **edu.unlv.mis768.labwork10.**

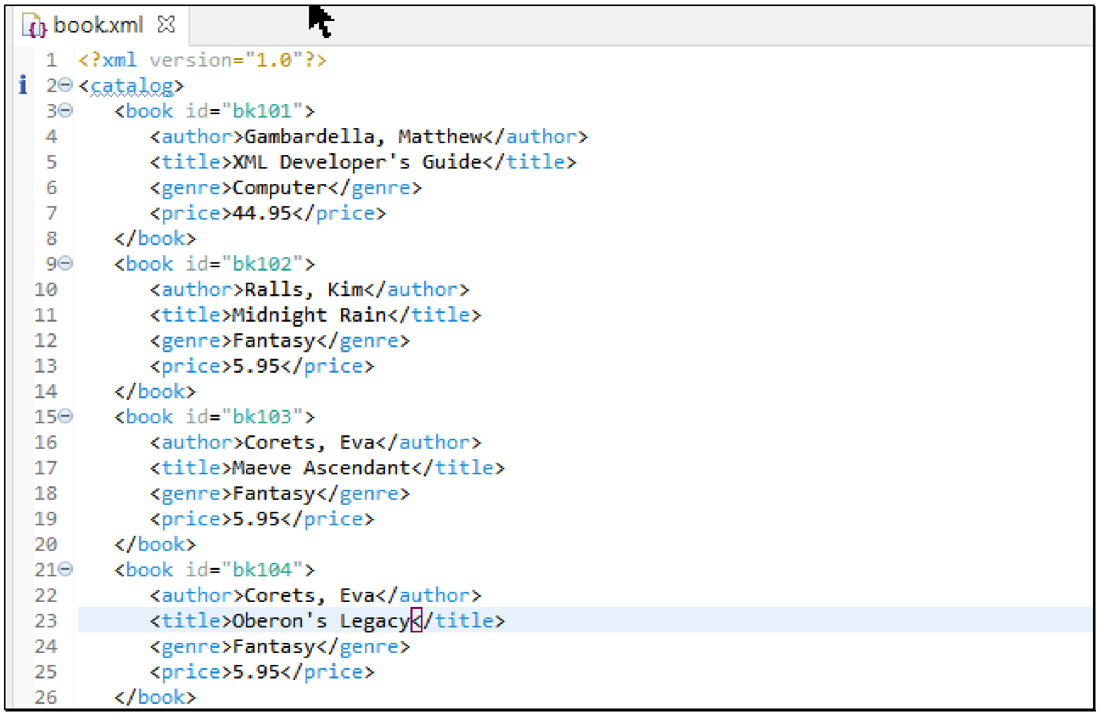
2. XML Data and Model Class

(4) Please move book.xml to the project directory if needed.
You can view the contain of book.xml in Eclipse by **right clicking \ Open With \ Generic Text Editor.**



```
<?xml version="1.0"?>
<catalog>
   <book id="bk101">
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
   </book>
   <book id="bk102">
      <author>Ralls, Kim</author>
      <title>Midnight Rain</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
   </book>
   <book id="bk103">
      <author>Corets, Eva</author>
      <title>Maeve Ascendant</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
   </book>
   <book id="bk104">
      <author>Corets, Eva</author>
h's Legacy</title>
sy</genre>
```

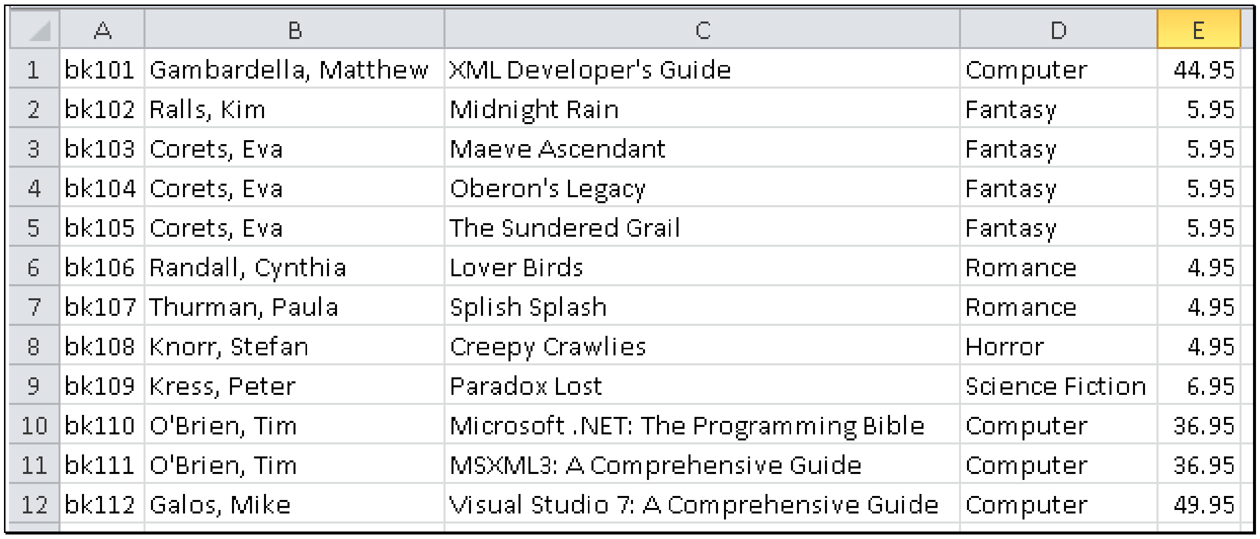
(5) The file book.xml contains the data for several books with different fields labeled in XML format.



book.xml

```
<?xml version="1.0"?>
<catalog>
   <book id="bk101">
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
   </book>
   <book id="bk102">
      <author>Ralls, Kim</author>
      <title>Midnight Rain</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
   </book>
   <book id="bk103">
      <author>Corets, Eva</author>
      <title>Maeve Ascendant</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
   </book>
   <book id="bk104">
      <author>Corets, Eva</author>
      <title>Oberon's Legacy</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
   </book>
```

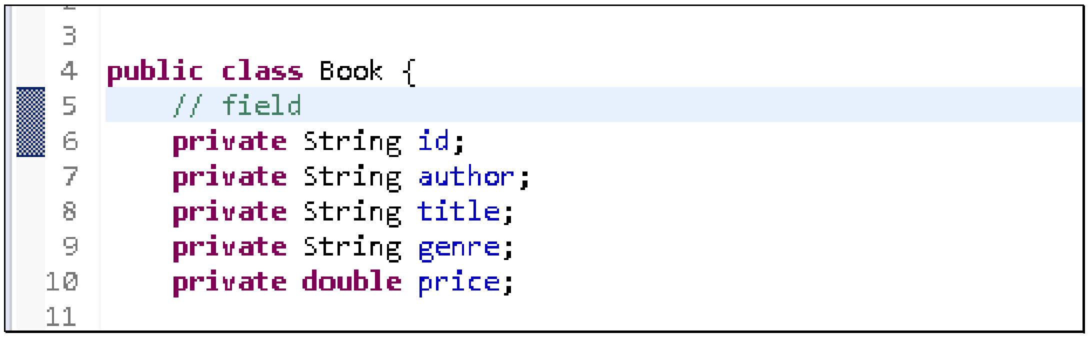
(6) Although it is useful to store the data in a structural way, we want to convert it into a tabular format for further analysis and manipulation.



	A	B	C	D	E
1	bk101	Gambardella, Matthew	XML Developer's Guide	Computer	44.95
2	bk102	Ralls, Kim	Midnight Rain	Fantasy	5.95
3	bk103	Corets, Eva	Maeve Ascendant	Fantasy	5.95
4	bk104	Corets, Eva	Oberon's Legacy	Fantasy	5.95
5	bk105	Corets, Eva	The Sundered Grail	Fantasy	5.95
6	bk106	Randall, Cynthia	Lover Birds	Romance	4.95
7	bk107	Thurman, Paula	Splish Splash	Romance	4.95
8	bk108	Knorr, Stefan	Creepy Crawlies	Horror	4.95
9	bk109	Kress, Peter	Paradox Lost	Science Fiction	6.95
10	bk110	O'Brien, Tim	Microsoft .NET: The Programming Bible	Computer	36.95
11	bk111	O'Brien, Tim	MSXML3: A Comprehensive Guide	Computer	36.95
12	bk112	Galos, Mike	Visual Studio 7: A Comprehensive Guide	Computer	49.95

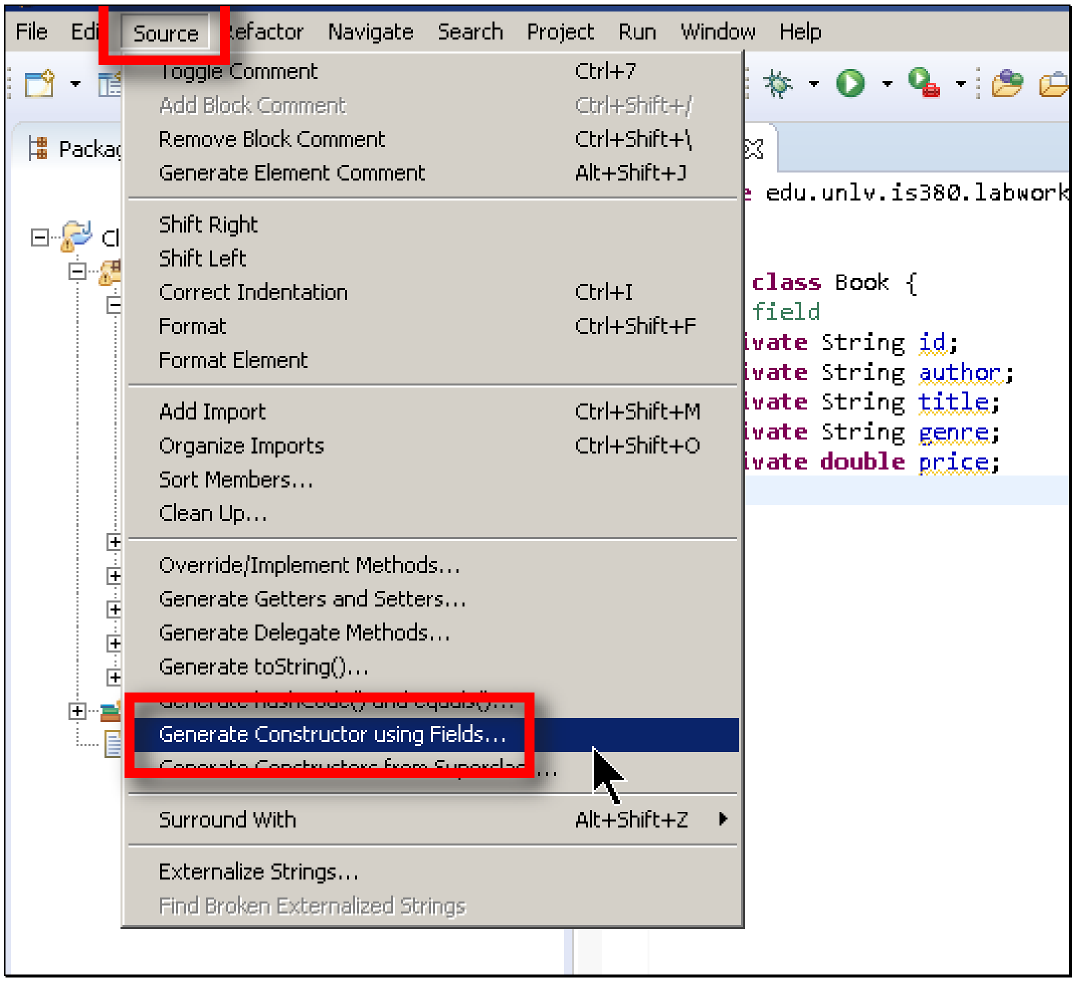
(7) We first define a class Book to represent the structure of the book data.

Please create a new class Book and declare the fields:



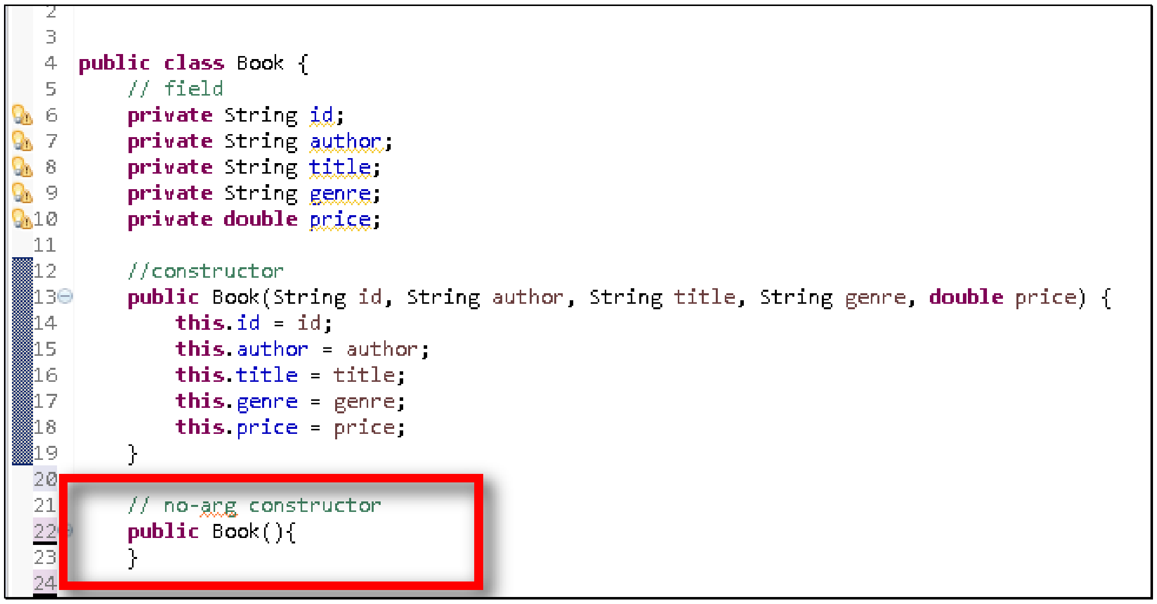
```

public class Book {
    // field
    private String id;
    private String author;
    private String title;
    private String genre;
    private double price;

```

(8) Create the constructors by selecting **Source \ Generate Constructor using Fields**.


File
Edit
Source
Refactor
Navigate
Search
Project
Run
Window
Help
Toggle Comment Ctrl+7
Add Block Comment Ctrl+Shift+/
Remove Block Comment Ctrl+Shift+\
Generate Element Comment Alt+Shift+J
Shift Right
Shift Left
Correct Indentation Ctrl+I
Format Ctrl+Shift+F
Format Element
Add Import Ctrl+Shift+M
Organize Imports Ctrl+Shift+O
Sort Members...
Clean Up...
Override/Implement Methods...
Generate Getters and Setters...
Generate Delegate Methods...
Generate toString()...
Generate hashCode() and equals()...
Generate Constructor using Fields...
Generate Constructors from Superclass...
Surround With Alt+Shift+Z
Externalize Strings...
Find Broken Externalized Strings

```
e edu.unlv.is380.labwork

 class Book {
 field
ivate String id;
ivate String author;
ivate String title;
ivate String genre;
ivate double price;
```

(9) Also create the no-arg constructor.



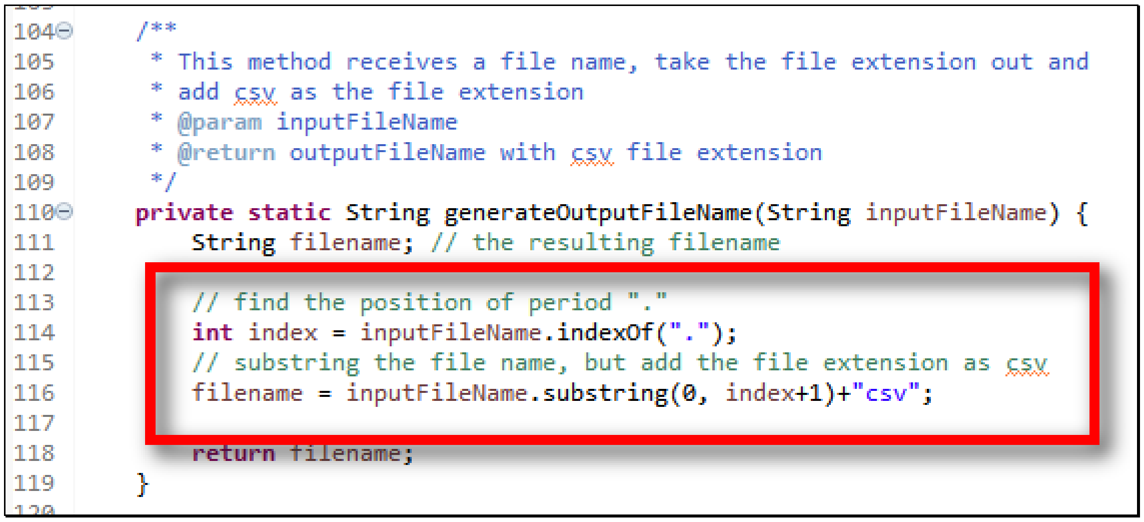
```
public class Book {
    // field
    private String id;
    private String author;
    private String title;
    private String genre;
    private double price;

    //constructor
    public Book(String id, String author, String title, String genre, double price) {
        this.id = id;
        this.author = author;
        this.title = title;
        this.genre = genre;
        this.price = price;
    }

    // no-arg constructor
    public Book(){
    }
```

(10) Also generate all the getters and setters.

(11) Please open the **BookDataFormatter** class, which is an application class that contains the main method.

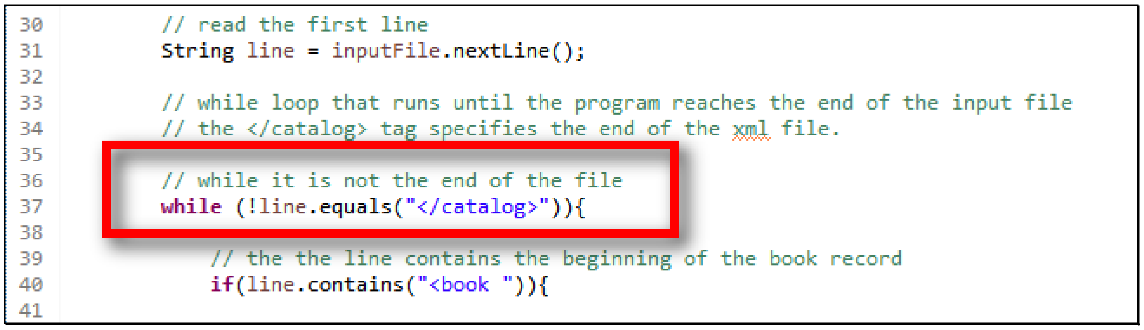
This program will ask the user to enter the file name, and create an output file with the same name but with the .csv file extension. Then process the file content.

Please complete the **generateOutputFileName()** method as following:



```
    /**
     * This method receives a file name, take the file extension out and
     * add csv as the file extension
     * @param inputFileName
     * @return outputFileName with csv file extension
     */
    private static String generateOutputFileName(String inputFileName) {
        String filename; // the resulting filename

        // find the position of period "."
        int index = inputFileName.indexOf(".");
        // substring the file name, but add the file extension as csv
        filename = inputFileName.substring(0, index+1)+"csv";

        return filename;
    }
```

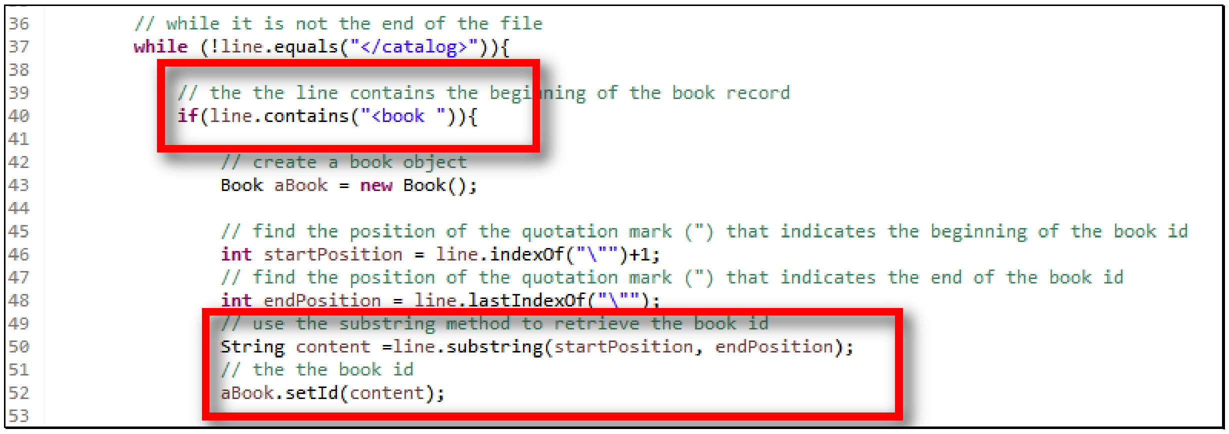
(12) Start with specifying the stopping condition of reading the file



```
        // read the first line
        String line = inputFile.nextLine();

        // while loop that runs until the program reaches the end of the input file
        // the </catalog> tag specifies the end of the xml file.

        // while it is not the end of the file
        while (!line.equals("</catalog>")){

            // the the line contains the beginning of the book record
            if(line.contains("<book ")){

```

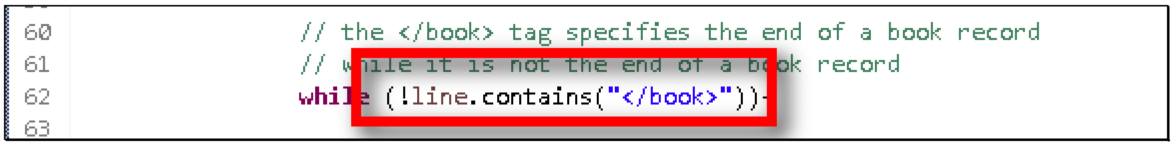
(13) Then find the start of a book record. The line would contain the tag **<book**

If a book record is located, retrieve the book id and set to the id field of the **Book** object.



```
        // while it is not the end of the file
        while (!line.equals("</catalog>")){

            // the the line contains the beginning of the book record
            if(line.contains("<book ")){

                // create a book object
                Book aBook = new Book();

                // find the position of the quotation mark (") that indicates the beginning of the book id
                int startPosition = line.indexOf("\"")+1;
                // find the position of the quotation mark (") that indicates the end of the book id
                int endPosition = line.lastIndexOf("\"");
                // use the substring method to retrieve the book id
                String content =line.substring(startPosition, endPosition);
                // the the book id
                aBook.setId(content);

```

(14) Then, read inside a book record to find each data field. The end of a book record is indicated by the tag </book>

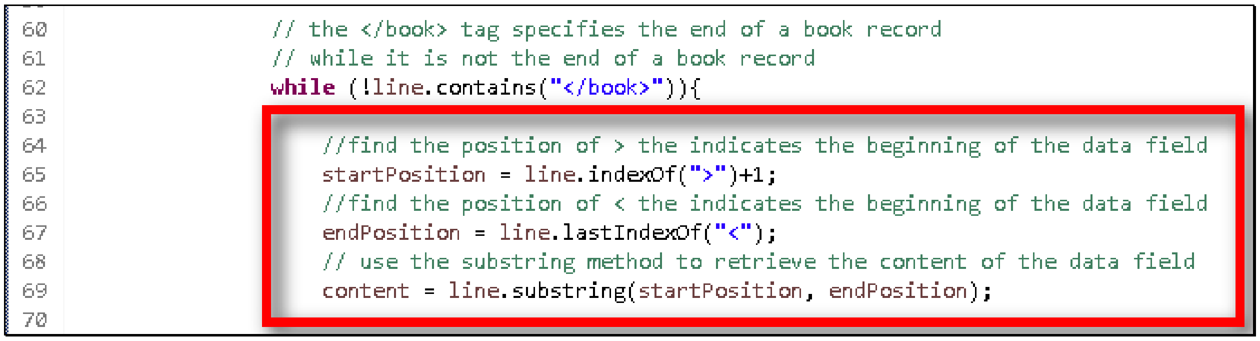


```
// the </book> tag specifies the end of a book record
// while it is not the end of a book record
while (!line.contains("</book>"))
```

(15) Within each data field, the content is specified by a pair of tags.

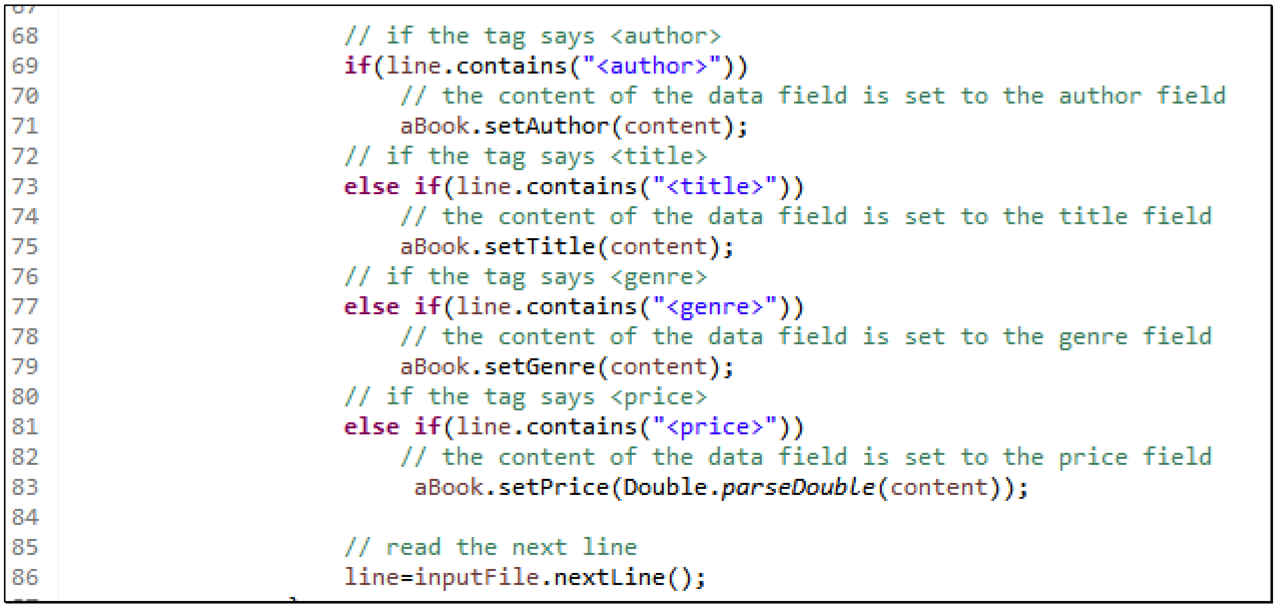
Find the starting position of the data content, and find the end position of the data content.

Retrieve the data content using the substring method.



```
// the </book> tag specifies the end of a book record
// while it is not the end of a book record
while (!line.contains("</book>")){

    //find the position of > the indicates the beginning of the data field
    startPosition = line.indexOf(">")+1;
    //find the position of < the indicates the beginning of the data field
    endPosition = line.lastIndexOf("<");
    // use the substring method to retrieve the content of the data field
    content = line.substring(startPosition, endPosition);
```

(16) Then specify the content to the data field according to the tag.

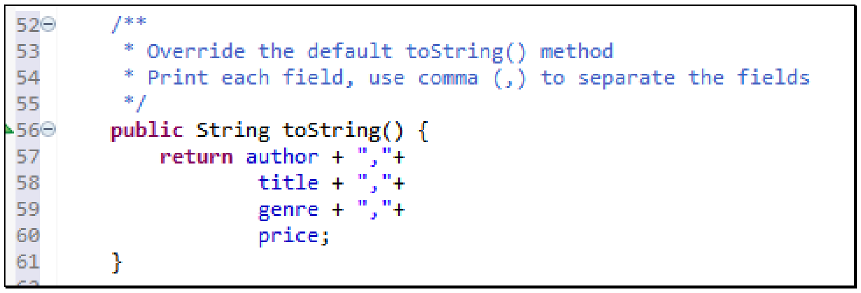


```
// if the tag says <author>
if(line.contains("<author>"))
    // the content of the data field is set to the author field
    aBook.setAuthor(content);
// if the tag says <title>
else if(line.contains("<title>"))
    // the content of the data field is set to the title field
    aBook.setTitle(content);
// if the tag says <genre>
else if(line.contains("<genre>"))
    // the content of the data field is set to the genre field
    aBook.setGenre(content);
// if the tag says <price>
else if(line.contains("<price>"))
    // the content of the data field is set to the price field
     aBook.setPrice(Double.parseDouble(content));

// read the next line
line=inputFile.nextLine();
```

(17) Now you can run and test the program. You will see 12 records are written to the csv file.

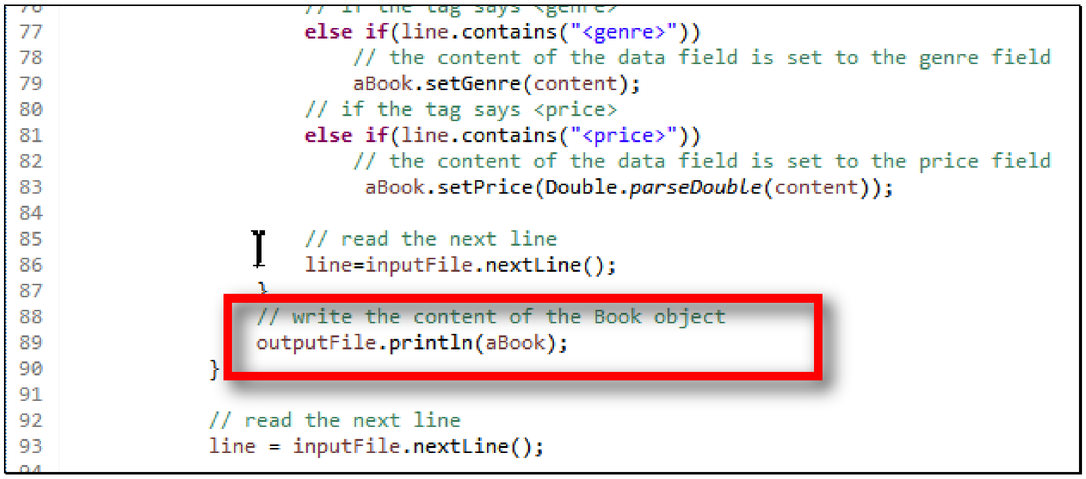
3. Implement toString() method in Book.jvava

(18) In the **Book** class, please implement a **toString()** method, that would concatenate the field into a single string, with comma (,) separating each field.



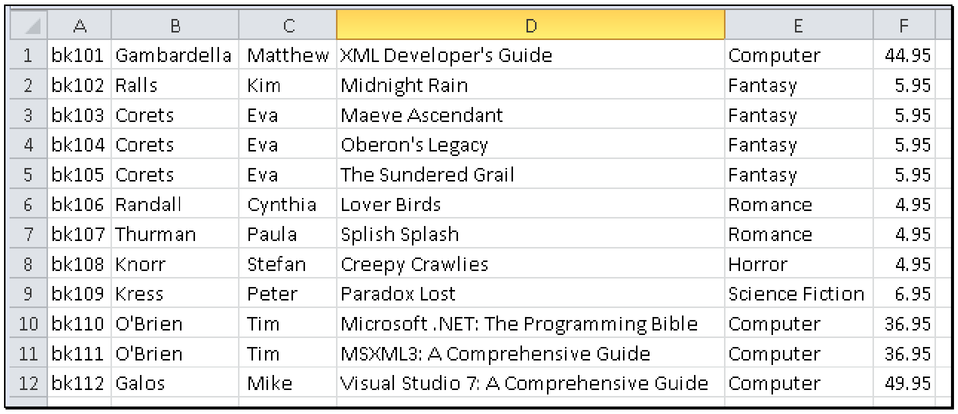
```
    /**
     * Override the default toString() method
     * Print each field, use comma (,) to separate the fields
     */
    public String toString() {
        return author + ","+
               title + ","+
               genre + ","+
               price;
    }
```

(19) Revise the **BookDataFormatter** program to use the **toString()** method in Book class, instead of printing each fields separately.

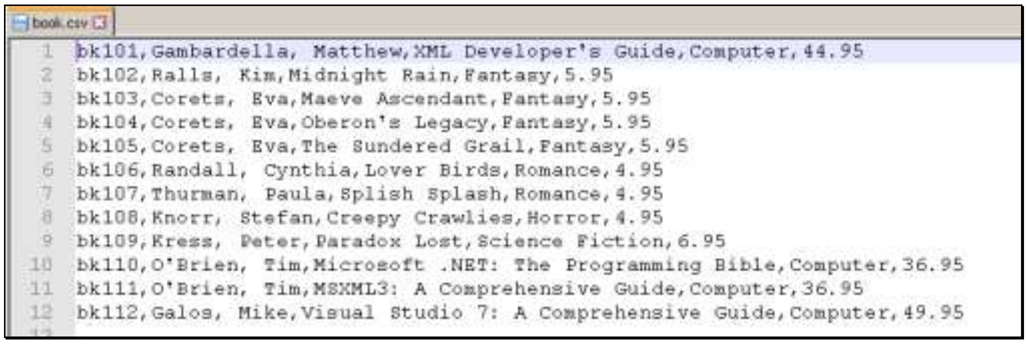


```
                    else if(line.contains("<genre>"))
                        // the content of the data field is set to the genre field
                        aBook.setGenre(content);
                    // if the tag says <price>
                    else if(line.contains("<price>"))
                        // the content of the data field is set to the price field
                         aBook.setPrice(Double.parseDouble(content));

                    // read the next line
                    line=inputFile.nextLine();
                }
                // write the content of the Book object
                outputFile.println(aBook);
            }

            // read the next line
            line = inputFile.nextLine();
```

(20) When you open the output file, you should see this:



	A	B	C	D	E	F
1	bk101	Gambardella	Matthew	XML Developer's Guide	Computer	44.95
2	bk102	Ralls	Kim	Midnight Rain	Fantasy	5.95
3	bk103	Corets	Eva	Maeve Ascendant	Fantasy	5.95
4	bk104	Corets	Eva	Oberon's Legacy	Fantasy	5.95
5	bk105	Corets	Eva	The Sundered Grail	Fantasy	5.95
6	bk106	Randall	Cynthia	Lover Birds	Romance	4.95
7	bk107	Thurman	Paula	Splish Splash	Romance	4.95
8	bk108	Knorr	Stefan	Creepy Crawlies	Horror	4.95
9	bk109	Kress	Peter	Paradox Lost	Science Fiction	6.95
10	bk110	O'Brien	Tim	Microsoft .NET: The Programming Bible	Computer	36.95
11	bk111	O'Brien	Tim	MSXML3: A Comprehensive Guide	Computer	36.95
12	bk112	Galos	Mike	Visual Studio 7: A Comprehensive Guide	Computer	49.95

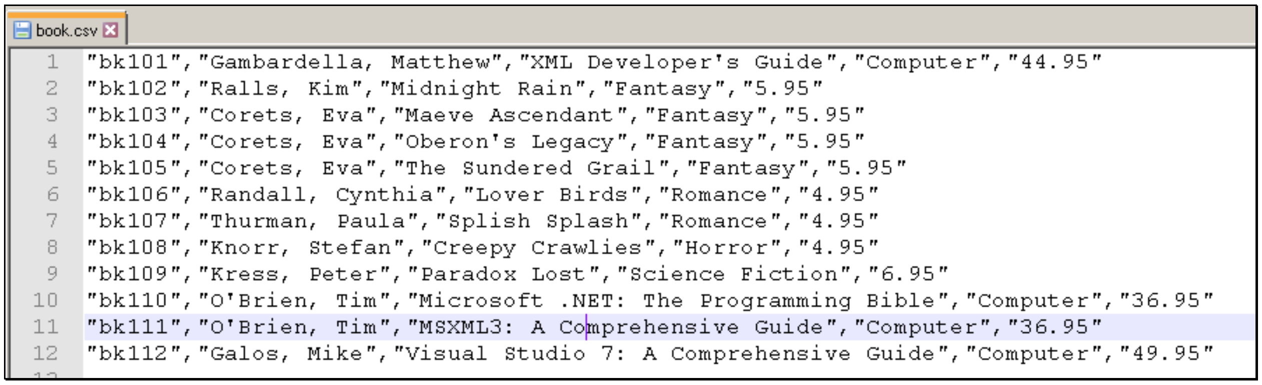


book.csv

```
bk101,Gambardella, Matthew,XML Developer's Guide,Computer,44.95
bk102,Ralls, Kim,Midnight Rain,Fantasy,5.95
bk103,Corets, Eva,Maeve Ascendant,Fantasy,5.95
bk104,Corets, Eva,Oberon's Legacy,Fantasy,5.95
bk105,Corets, Eva,The Sundered Grail,Fantasy,5.95
bk106,Randall, Cynthia,Lover Birds,Romance,4.95
bk107,Thurman, Paula,Splish Splash,Romance,4.95
bk108,Knorr, Stefan,Creepy Crawlies,Horror,4.95
bk109,Kress, Peter,Paradox Lost,Science Fiction,6.95
bk110,O'Brien, Tim,Microsoft .NET: The Programming Bible,Computer,36.95
bk111,O'Brien, Tim,MSXML3: A Comprehensive Guide,Computer,36.95
bk112,Galos, Mike,Visual Studio 7: A Comprehensive Guide,Computer,49.95
```

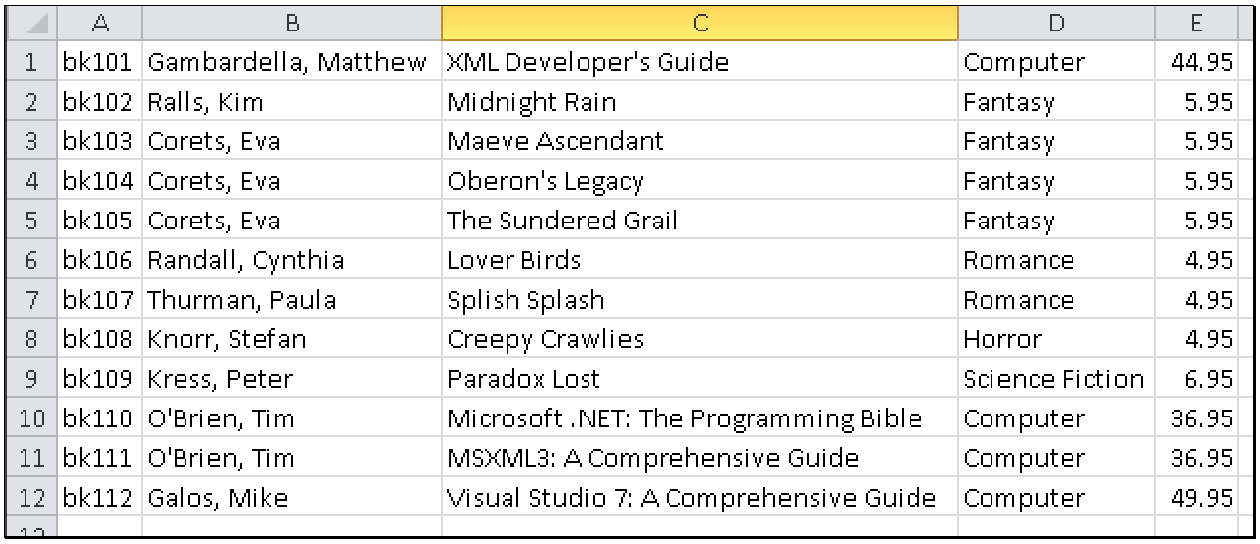
(21) Please note that, the **author** field contains a comma (,) and therefore is treated as two separate columns when you open it in Excel.

To fix this, we can add quotation marks (" ") around one field, to make it look like the following:



```
"bk101","Gambardella, Matthew","XML Developer's Guide","Computer","44.95"
"bk102","Ralls, Kim","Midnight Rain","Fantasy","5.95"
"bk103","Corets, Eva","Maeve Ascendant","Fantasy","5.95"
"bk104","Corets, Eva","Oberon's Legacy","Fantasy","5.95"
"bk105","Corets, Eva","The Sundered Grail","Fantasy","5.95"
"bk106","Randall, Cynthia","Lover Birds","Romance","4.95"
"bk107","Thurman, Paula","Splish Splash","Romance","4.95"
"bk108","Knorr, Stefan","Creepy Crawlies","Horror","4.95"
"bk109","Kress, Peter","Paradox Lost","Science Fiction","6.95"
"bk110","O'Brien, Tim","Microsoft .NET: The Programming Bible","Computer","36.95"
"bk111","O'Brien, Tim","MSXML3: A Comprehensive Guide","Computer","36.95"
"bk112","Galos, Mike","Visual Studio 7: A Comprehensive Guide","Computer","49.95"
```

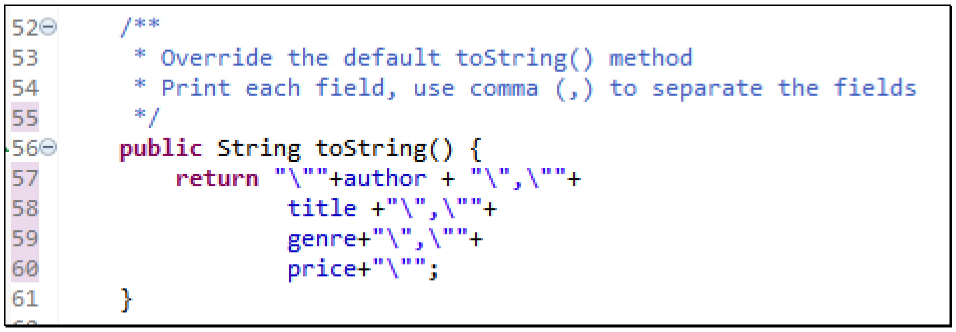
So that when you open it in excel, the **author** field will be treated as one column.



	A	B	C	D	E
1	bk101	Gambardella, Matthew	XML Developer's Guide	Computer	44.95
2	bk102	Ralls, Kim	Midnight Rain	Fantasy	5.95
3	bk103	Corets, Eva	Maeve Ascendant	Fantasy	5.95
4	bk104	Corets, Eva	Oberon's Legacy	Fantasy	5.95
5	bk105	Corets, Eva	The Sundered Grail	Fantasy	5.95
6	bk106	Randall, Cynthia	Lover Birds	Romance	4.95
7	bk107	Thurman, Paula	Splish Splash	Romance	4.95
8	bk108	Knorr, Stefan	Creepy Crawlies	Horror	4.95
9	bk109	Kress, Peter	Paradox Lost	Science Fiction	6.95
10	bk110	O'Brien, Tim	Microsoft .NET: The Programming Bible	Computer	36.95
11	bk111	O'Brien, Tim	MSXML3: A Comprehensive Guide	Computer	36.95
12	bk112	Galos, Mike	Visual Studio 7: A Comprehensive Guide	Computer	49.95

(22) Quotation mark is a special character in Java, in order to use it in a String, you need to add a backslash (\) to it.

For example, the **toString()** method can be revised as following:

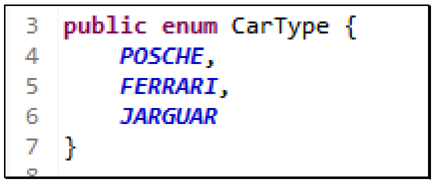


```java
    /**
     * Override the default toString() method
     * Print each field, use comma (,) to separate the fields
     */
    public String toString() {
        return "\""+author + "\",\""+
                title +"\",\""+
                genre+"\",\""+
                price+"\"";
    }
```

4. Enumerated Types

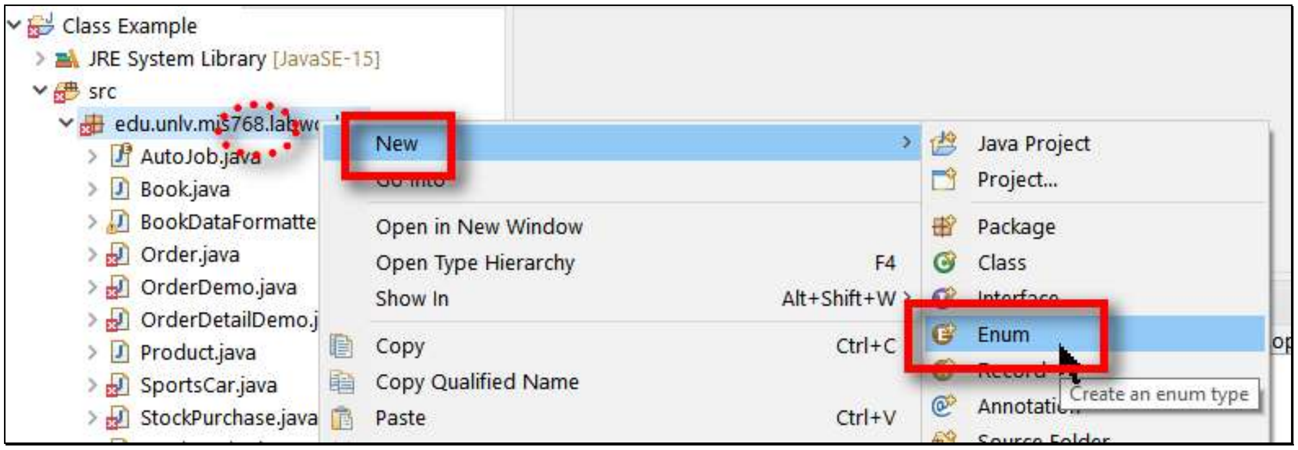
(23) This example illustrates the usage of enum in a program.

Please open **CarType.java**. It is a Enum type and has three values defined:

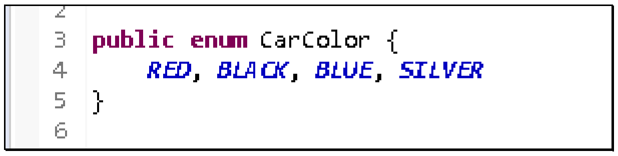


```java
public enum CarType {
    POSCHE,
    FERRARI,
    JARGUAR
}
```

(24) Create a **Enum**. Right click on the package, select **New** and then **Enum**.

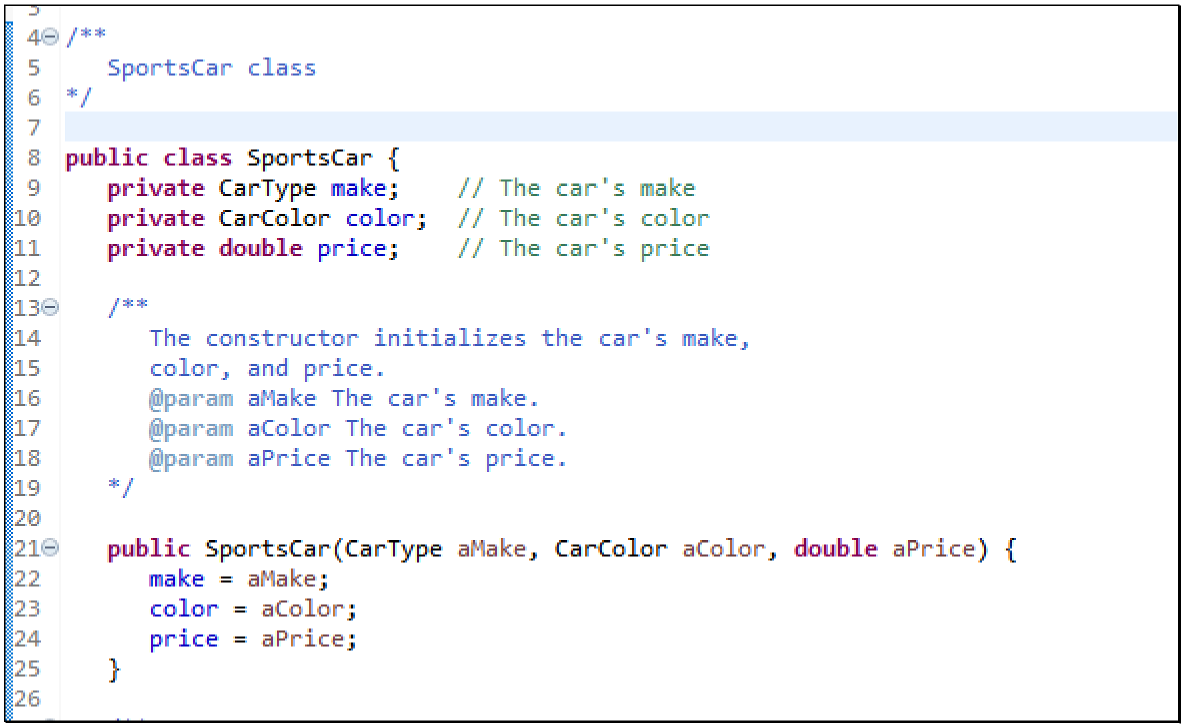


(25) Name it as **CarColor** with the following code:



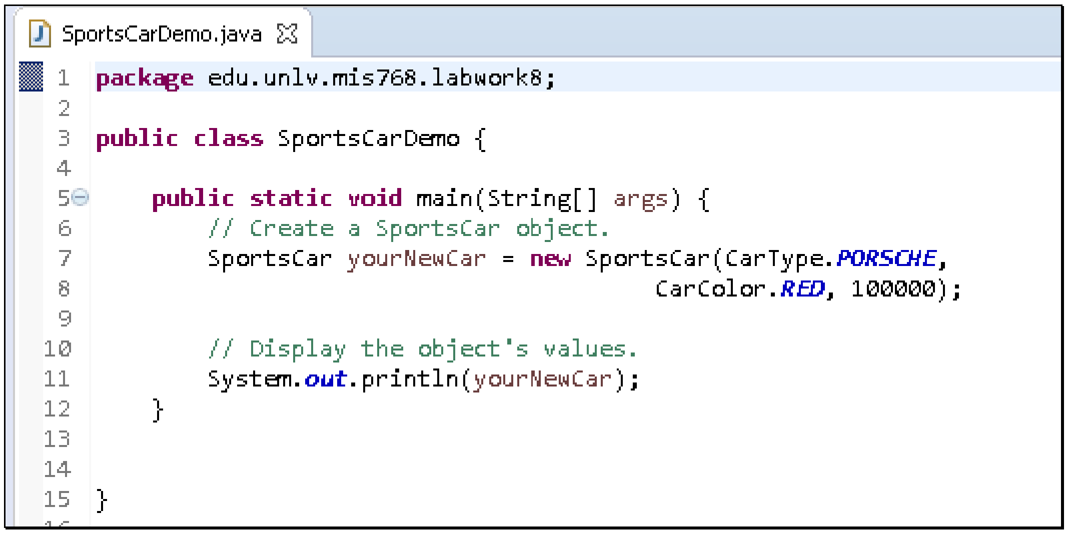
```java
public enum CarColor {
    RED, BLACK, BLUE, SILVER
}
```

(26) Open **SportsCar.java** in the editor to review the code. The first two fields are defined using **CarType** and **CarColor** as the data type.



```
/**
   SportsCar class
*/

public class SportsCar {
   private CarType make;     // The car's make
   private CarColor color;   // The car's color
   private double price;     // The car's price

   /**
      The constructor initializes the car's make,
      color, and price.
      @param aMake The car's make.
      @param aColor The car's color.
      @param aPrice The car's price.
   */

   public SportsCar(CarType aMake, CarColor aColor, double aPrice) {
      make = aMake;
      color = aColor;
      price = aPrice;
   }

```

(27) Create **SportsCarDemo** program with the following code to instantiate an SpartsCar object. You will see that the value of first field is limited to the values defined in CarType. Similarly, the value of the second field is lmited to the values of CarColor.



SportsCarDemo.java

```
package edu.unlv.mis768.labwork8;

public class SportsCarDemo {

    public static void main(String[] args) {
        // Create a SportsCar object.
        SportsCar yourNewCar = new SportsCar(CarType.PORSCHE,
                                        CarColor.RED, 100000);

        // Display the object's values.
        System.out.println(yourNewCar);
    }


}
```

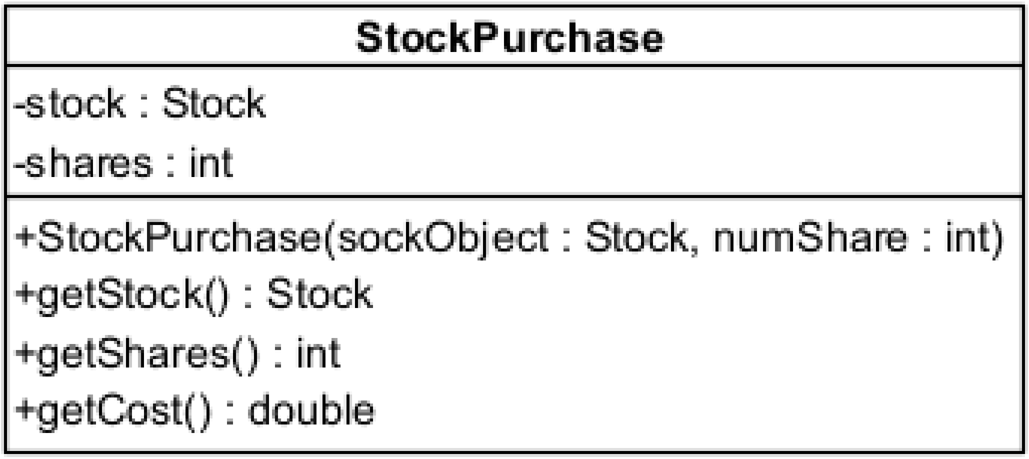
(28) You can run **SportsCarDemo.java** to see how it works.

5. Enum with associated values

(29) Open **AutoJob**.java. This **Enum** type declare different types of jobs that can be complete in an auto shop, with the price with each job assigned.
We will see this example later.

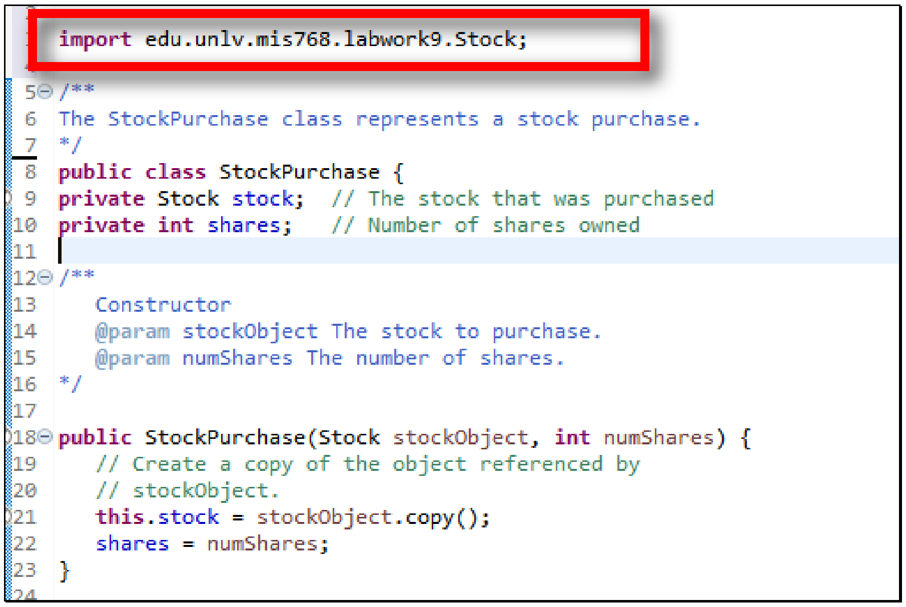
6. Class Collaboration

(30) Open **StockPurchase.java** in the editor to review the code.



StockPurchase
-stock : Stock -shares : int
+StockPurchase(sockObject : Stock, numShare : int) +getStock() : Stock +getShares() : int +getCost() : double

(31) This class needs the Stock class. Please add the import statement as shown below:



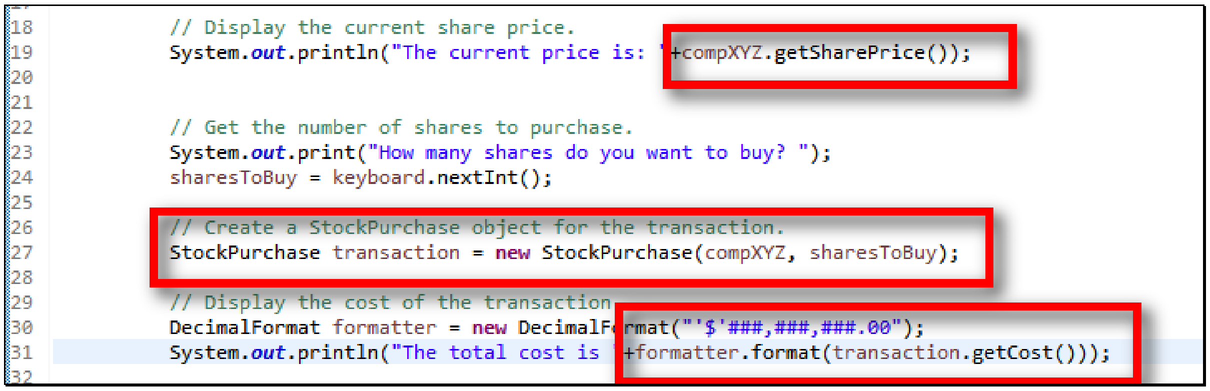
```
import edu.unlv.mis768.labwork9.Stock;

/**
The StockPurchase class represents a stock purchase.
*/
public class StockPurchase {
private Stock stock;  // The stock that was purchased
private int shares;   // Number of shares owned

/**
   Constructor
   @param stockObject The stock to purchase.
   @param numShares The number of shares.
*/

public StockPurchase(Stock stockObject, int numShares) {
   // Create a copy of the object referenced by
   // stockObject.
   this.stock = stockObject.copy();
   shares = numShares;
}
```

(32) Please open **StockTrader.java.**

It instantiate a **`Stock`** object and display the current share price of **compXYZ**. It ask the user how many shares to purchase.

It then instantiate a **StockPurchase** object, with the fields initialized (compXYZ & shareToBuy) and then use the method of **StockPurchase** to show the total cost.

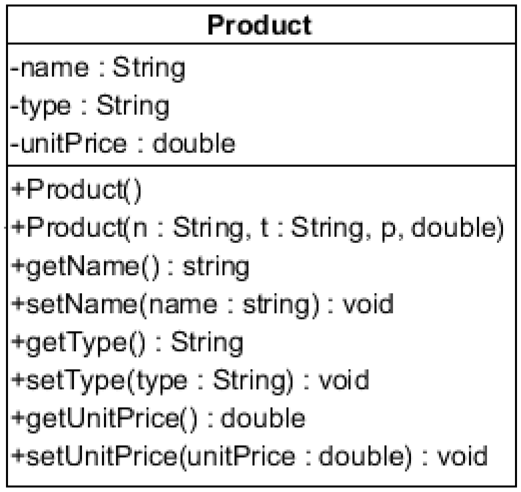
(33) Please complete the following program:



```
        // Display the current share price.
        System.out.println("The current price is: "+compXYZ.getSharePrice());


        // Get the number of shares to purchase.
        System.out.print("How many shares do you want to buy? ");
        sharesToBuy = keyboard.nextInt();

        // Create a StockPurchase object for the transaction.
        StockPurchase transaction = new StockPurchase(compXYZ, sharesToBuy);

        // Display the cost of the transaction
        DecimalFormat formatter = new DecimalFormat("'$'###,###,###.00");
        System.out.println("The total cost is "+formatter.format(transaction.getCost()));
```

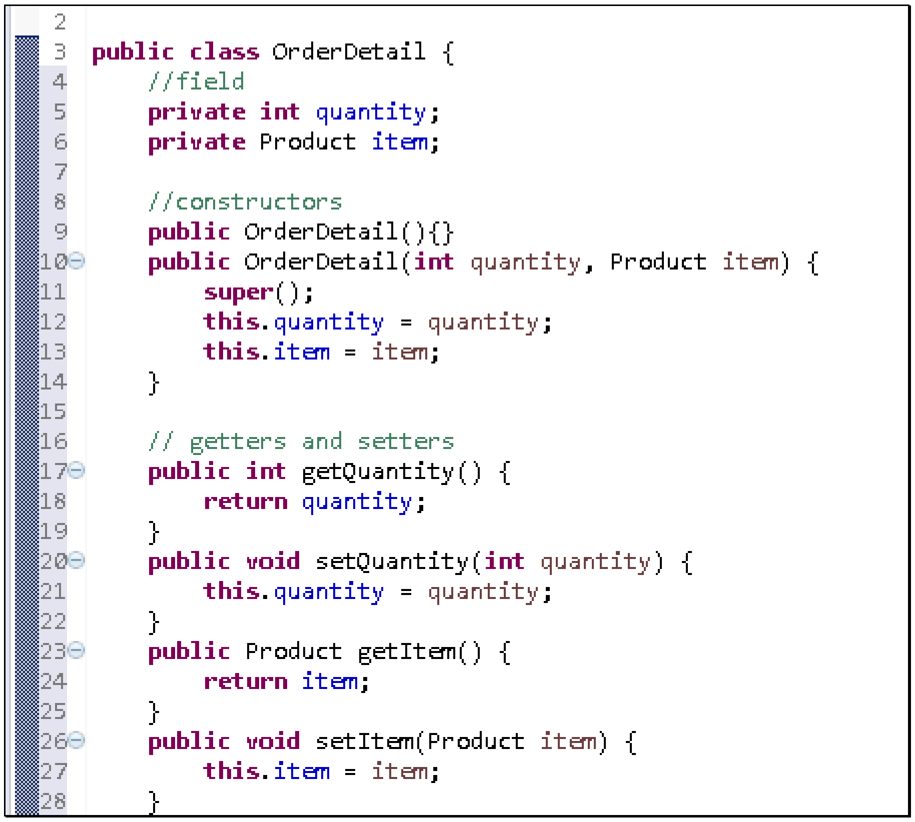
7. Class Collaboration: Example (2)

(34) The **Product** class is created based on the class diagram.



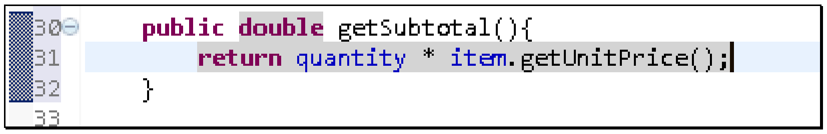
Product
-name : String -type : String -unitPrice : double
+Product() +Product(n : String, t : String, p, double) +getName() : string +setName(name : string) : void +getType() : String +setType(type : String) : void +getUnitPrice() : double +setUnitPrice(unitPrice : double) : void

(35) Please create **OrderDetail** class and define the fields, constructors, and getters and setters. **OrderDetail** is a class with an objects as a field.



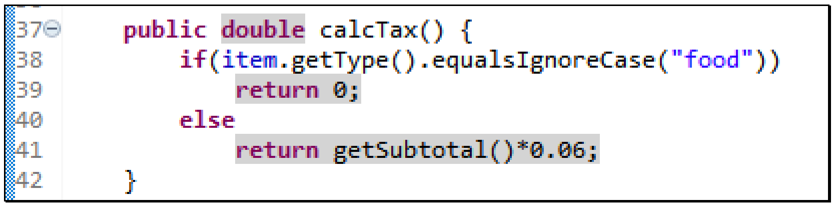
```
public class OrderDetail {
    //field
    private int quantity;
    private Product item;

    //constructors
    public OrderDetail(){}
    public OrderDetail(int quantity, Product item) {
        super();
        this.quantity = quantity;
        this.item = item;
    }

    // getters and setters
    public int getQuantity() {
        return quantity;
    }
    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
    public Product getItem() {
        return item;
    }
    public void setItem(Product item) {
        this.item = item;
    }
```

(36) The subtotal is calculated based on the quantity and the unit price of the product. Implement **getSubtatal()** as following:



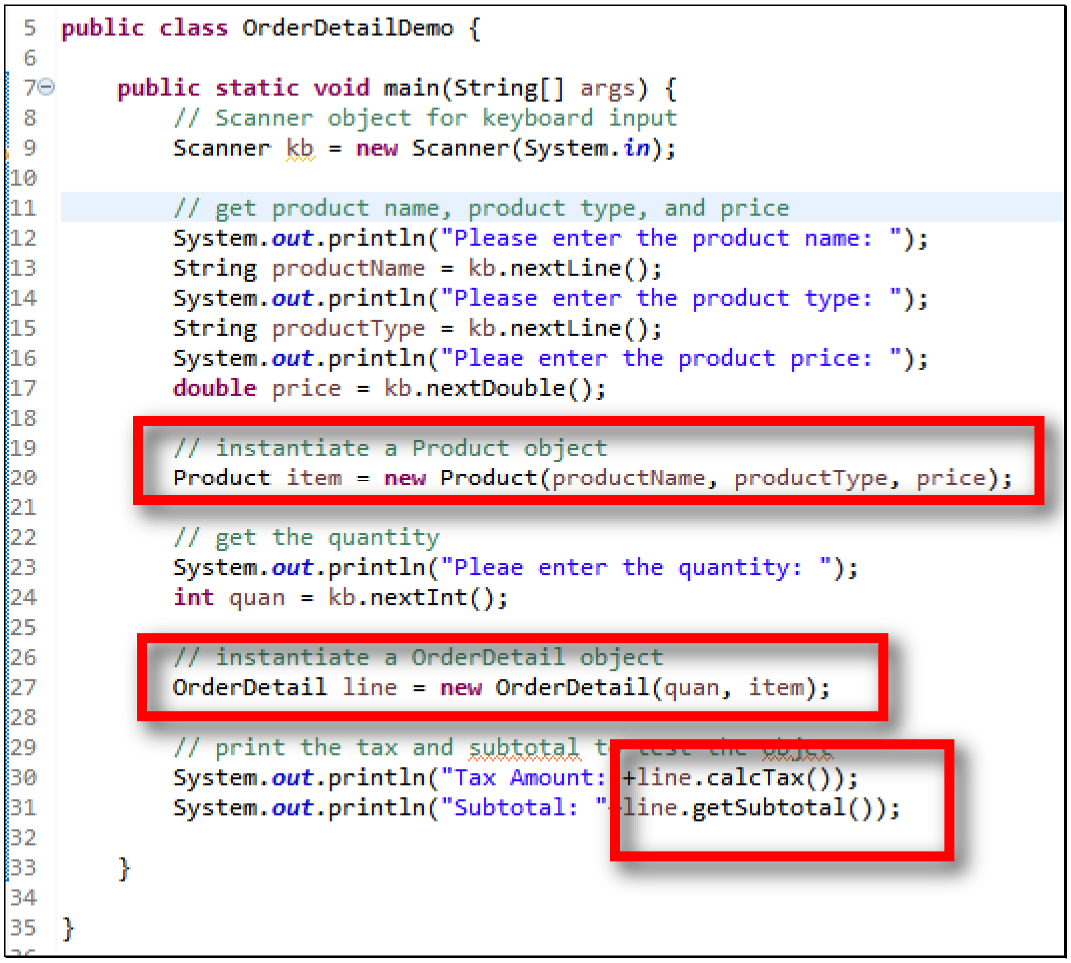
```
    public double getSubtotal(){
        return quantity * item.getUnitPrice();
    }
```

(37) When the product type is "food," the tax rate is 0, and it is 0.06 for all other products. We can implement the **calcTax()** method as following:



```
    public double calcTax() {
        if(item.getType().equalsIgnoreCase("food"))
            return 0;
        else
            return getSubtotal()*0.06;
    }
```

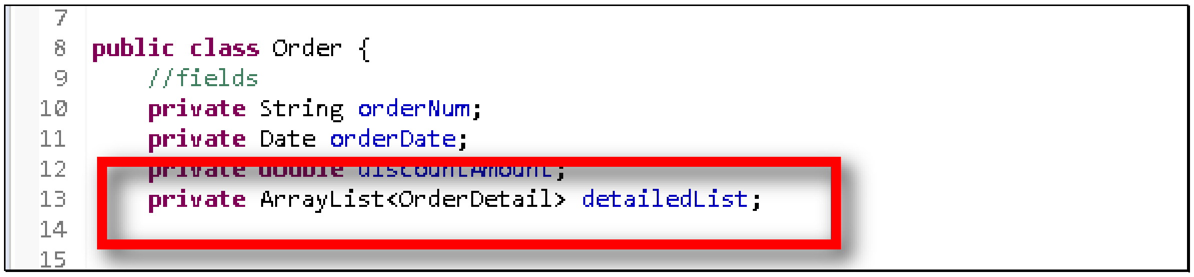
(38) We can test the two classes using the **OrderDetailDemo** program. Please complete the program:



```java
public class OrderDetailDemo {

    public static void main(String[] args) {
        // Scanner object for keyboard input
        Scanner kb = new Scanner(System.in);

        // get product name, product type, and price
        System.out.println("Please enter the product name: ");
        String productName = kb.nextLine();
        System.out.println("Please enter the product type: ");
        String productType = kb.nextLine();
        System.out.println("Pleae enter the product price: ");
        double price = kb.nextDouble();

        // instantiate a Product object
        Product item = new Product(productName, productType, price);

        // get the quantity
        System.out.println("Pleae enter the quantity: ");
        int quan = kb.nextInt();

        // instantiate a OrderDetail object
        OrderDetail line = new OrderDetail(quan, item);

        // print the tax and subtotal to test the object
        System.out.println("Tax Amount: "+line.calcTax());
        System.out.println("Subtotal: "+line.getSubtotal());

    }

}
```

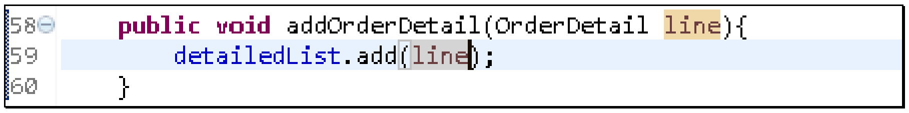
8. Class Aggregation

(39) **Order** Class is partially completed. Please add a new field **detailedList** to the class definition, and also create the getter and setter for it.



```java

public class Order {
    //fields
    private String orderNum;
    private Date orderDate;
    private double discountAmount;
    private ArrayList<OrderDetail> detailedList;


```

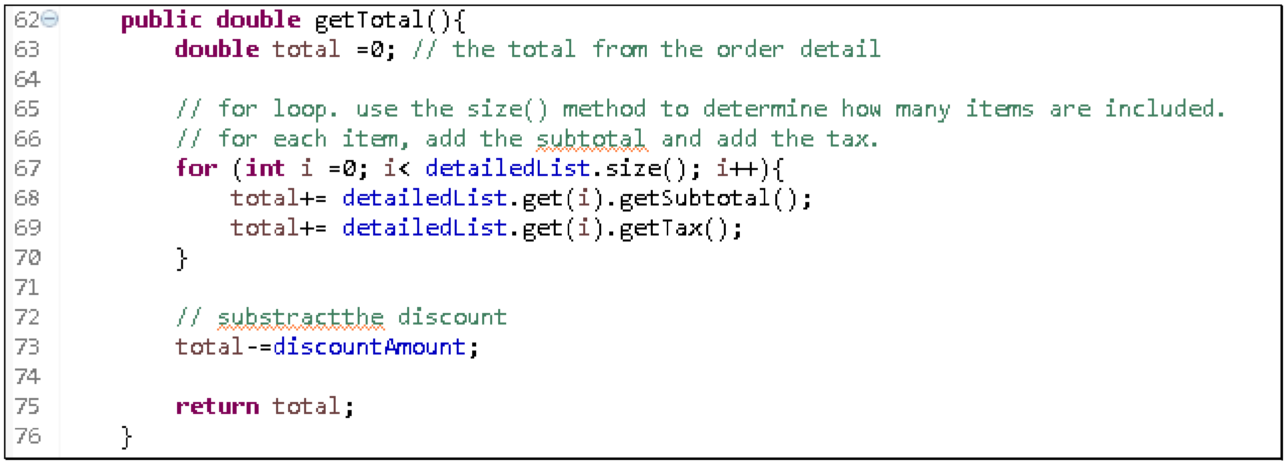
(40) Please add the **addOrderDetail()** method that allows adding a line of order detail at a time.



```java
    public void addOrderDetail(OrderDetail line){
        detailedList.add(line);
    }
```

(41) Also add the **getTotal()** method and use a for loop to traverse through the **ArrayList**.

In the loop, get each line item, use the **getSubtotal**() and **getTax**() method to calculate the total for this order.

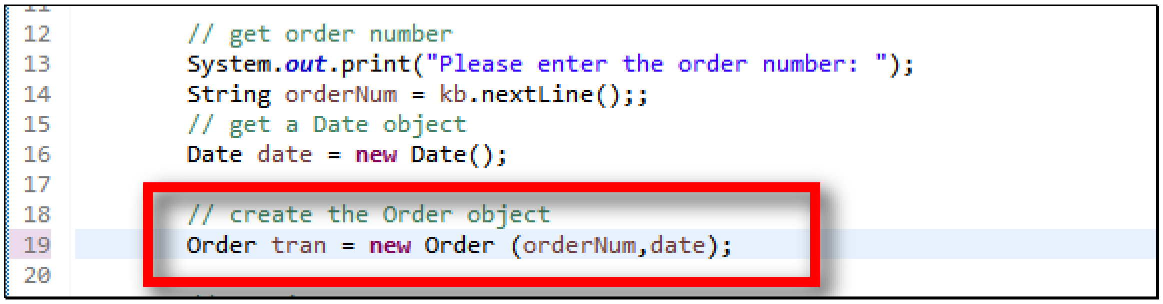


```
    public double getTotal(){
        double total =0; // the total from the order detail

        // for loop. use the size() method to determine how many items are included.
        // for each item, add the subtotal and add the tax.
        for (int i =0; i< detailedList.size(); i++){
            total+= detailedList.get(i).getSubtotal();
            total+= detailedList.get(i).getTax();
        }

        // substractthe discount
        total-=discountAmount;

        return total;
    }
```

(42) We can test the orders using the **OrderDemo** program.

In this application, we need to first get the order number from the user, and use the system date as the order date.

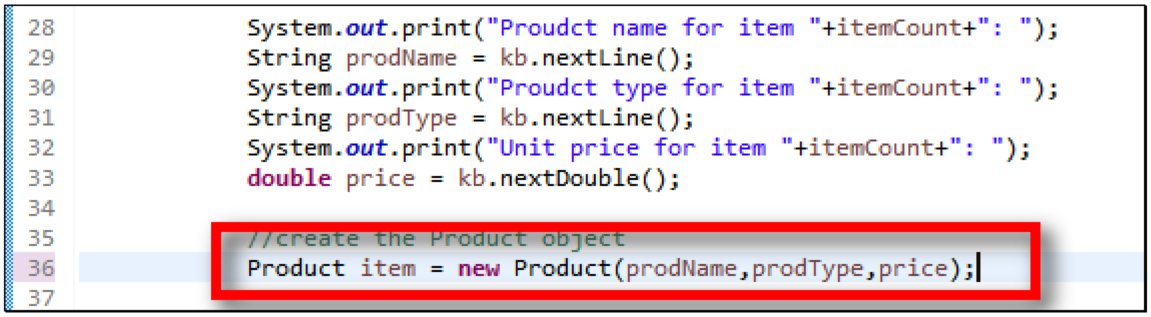
Please create an **Order** object based on order number and date.



```
        // get order number
        System.out.print("Please enter the order number: ");
        String orderNum = kb.nextLine();;
        // get a Date object
        Date date = new Date();

        // create the Order object
        Order tran = new Order (orderNum,date);

```

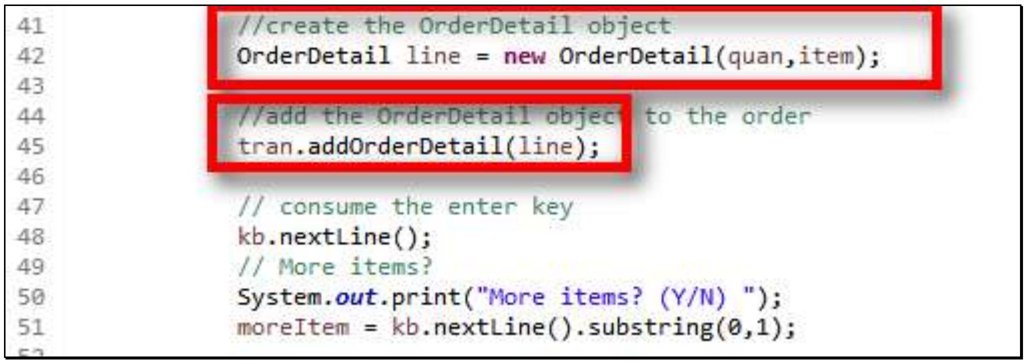
(43) Then, use a while loop to get all the order lines with product and quantity.

After the product name, type, and price are entered, create a **Product** object.



```
            System.out.print("Proudct name for item "+itemCount+": ");
            String prodName = kb.nextLine();
            System.out.print("Proudct type for item "+itemCount+": ");
            String prodType = kb.nextLine();
            System.out.print("Unit price for item "+itemCount+": ");
            double price = kb.nextDouble();

            //create the Product object
            Product item = new Product(prodName,prodType,price);

```

(44) Then get the quantity. Once we have the quantity and the product item, create the **OrderDetail** object. The **OrderDetail** object is a line in the **Order**, and should be added to the **detailedList** for the **Order**.



```
41              //create the OrderDetail object
42              OrderDetail line = new OrderDetail(quan,item);
43
44              //add the OrderDetail object to the order
45              tran.addOrderDetail(line);
46
47              // consume the enter key
48              kb.nextLine();
49              // More items?
50              System.out.print("More items? (Y/N) ");
51              moreItem = kb.nextLine().substring(0,1);
```

(45) In printing the receipt, we use a **for** loop to go through the **ArrayList**.

Use the **getDetailedList**() method to get the **ArrayList**.

Use the **get**() method to get each item in the **ArrayList** (i.e., an **OrderDetail** object).

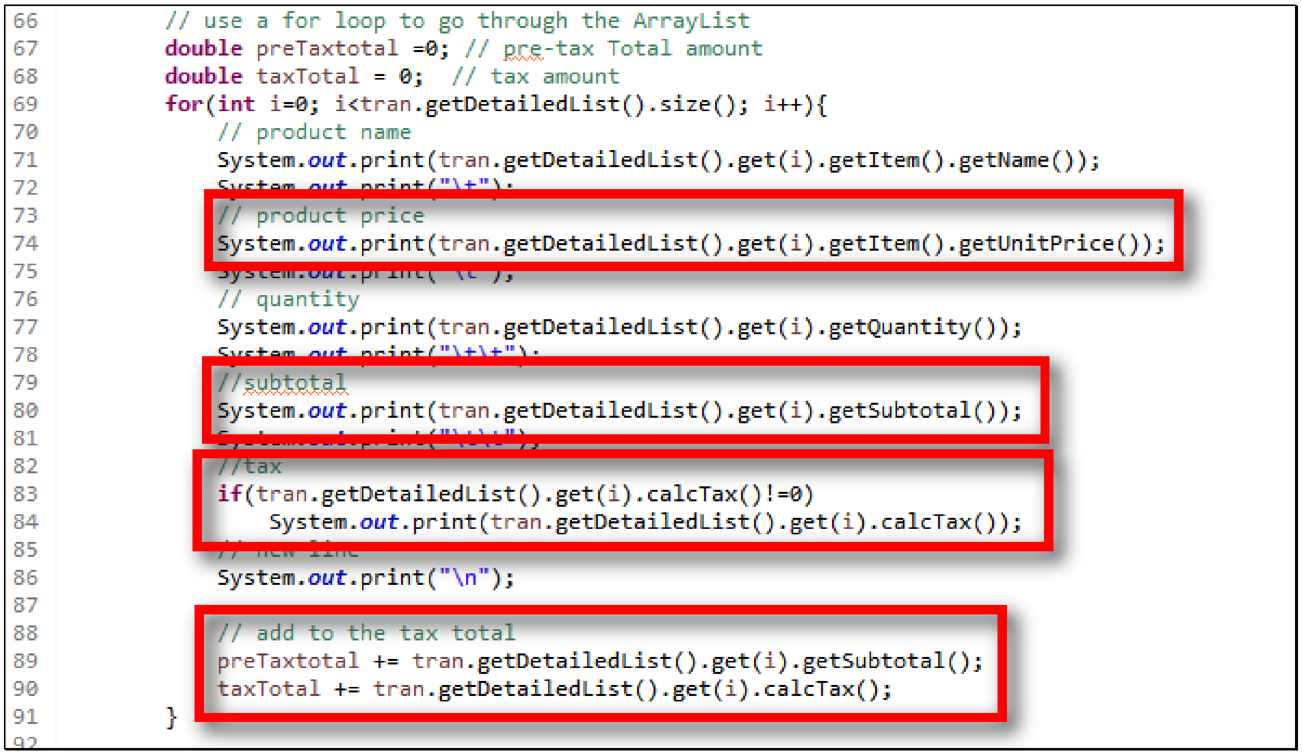
For each **OrderDetail** object, use **getItem**() method to get the **Product** object.

For each **Product** object, use **getName**() method to get the product name.

At line 77, please retrieve the product price for the **Product** object.

At line 83, please get the subtotal of the **OrderDetail** object. (also add it to the total at line 92)

At line 87, get the tax of the **OrderDetail** object. (also add it to the tax total at line 93.)



```
66          // use a for loop to go through the ArrayList
67          double preTaxtotal =0; // pre-tax Total amount
68          double taxTotal = 0;  // tax amount
69          for(int i=0; i<tran.getDetailedList().size(); i++){
70              // product name
71              System.out.print(tran.getDetailedList().get(i).getItem().getName());
72              System.out.print("\t");
73              // product price
74              System.out.print(tran.getDetailedList().get(i).getItem().getUnitPrice());
75              System.out.print("\t");
76              // quantity
77              System.out.print(tran.getDetailedList().get(i).getQuantity());
78              System.out.print("\t\t");
79              //subtotal
80              System.out.print(tran.getDetailedList().get(i).getSubtotal());
81              System.out.print("\t\t");
82              //tax
83              if(tran.getDetailedList().get(i).calcTax()!=0)
84                  System.out.print(tran.getDetailedList().get(i).calcTax());
85              // new line
86              System.out.print("\n");
87
88              // add to the tax total
89              preTaxtotal += tran.getDetailedList().get(i).getSubtotal();
90              taxTotal += tran.getDetailedList().get(i).calcTax();
91          }
```

(46) You can now run and test the program.