Classes (3)

Han-fen Hu

Tony Gaddis (2019) Starting Out with Java: From Control Structures through Data Structures, 4th Edition

UNLV

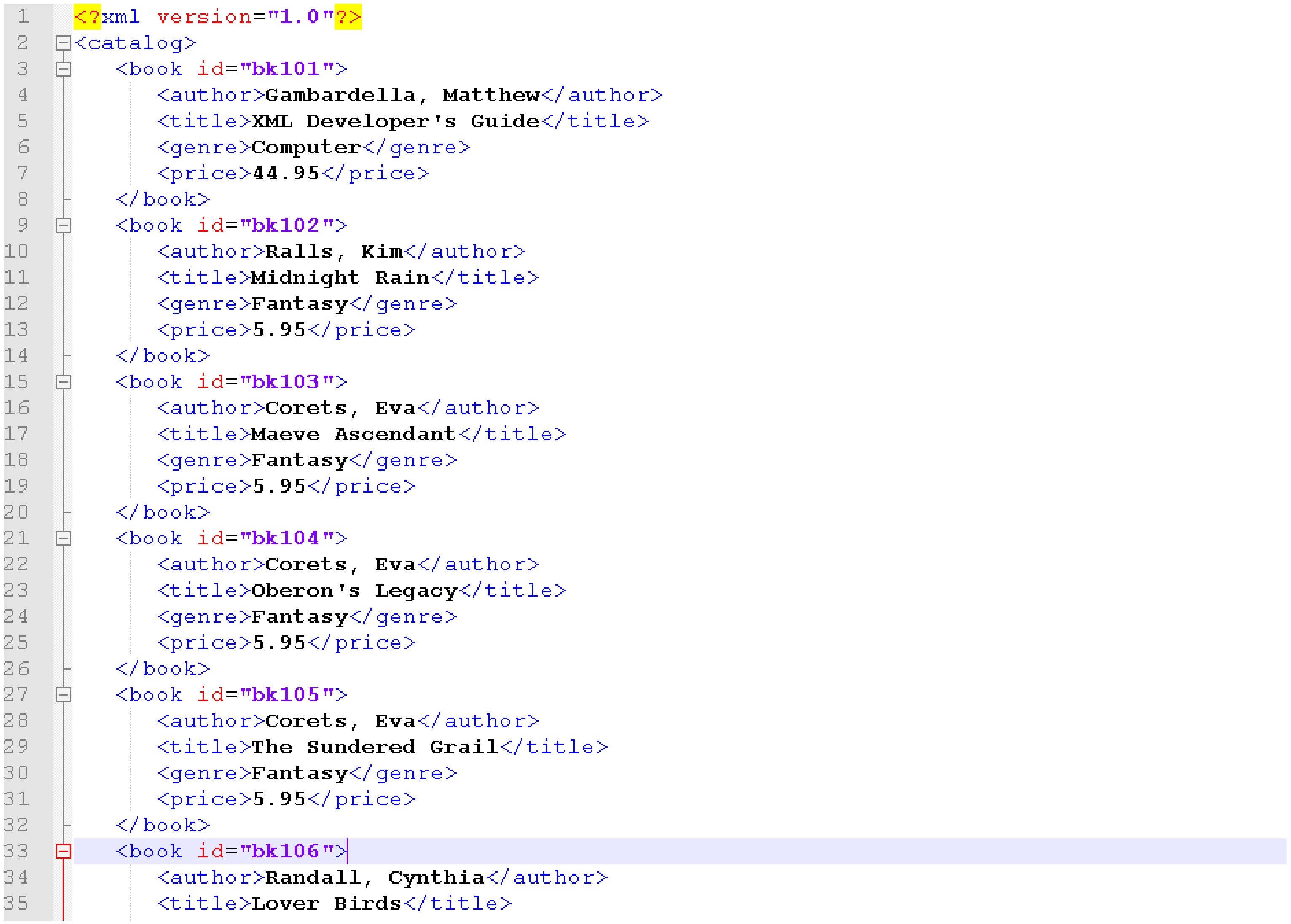
Outline

- XML and Class
- Enumerated Types
- Class Collaboration
- Class Aggregation

XML and Class

- XML
  - Stands for eXtensible Markup Language
  - XML was designed to store and transport data
- We can create model classes corresponding to the structure of the XML file to store the data

XML File example



```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>
    <book id="bk105">
        <author>Corets, Eva</author>
        <title>The Sundered Grail</title>
        <genre>Fantasy</genre>
        <price>5.95</price>
    </book>
    <book id="bk106">
        <author>Randall, Cynthia</author>
        <title>Lover Birds</title>
```

Lab (1)

- Book.java
  - A model class the corresponding to the data structure of the XML file
- BookDataFormatter.java
  - An application class that asks 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.

String Processing (1)

- generateOutputFileName()

inputFileName

b	o	o	k	.	x	m	l
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

index = 4

```
// 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";
```

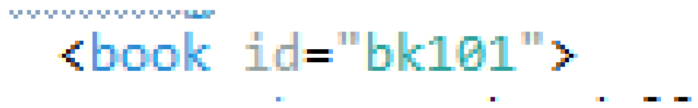
Add “csv to the string
filename = “book.” + “csv”

Substring, starting from position 0, get 5 characters

b	o	o	k	.

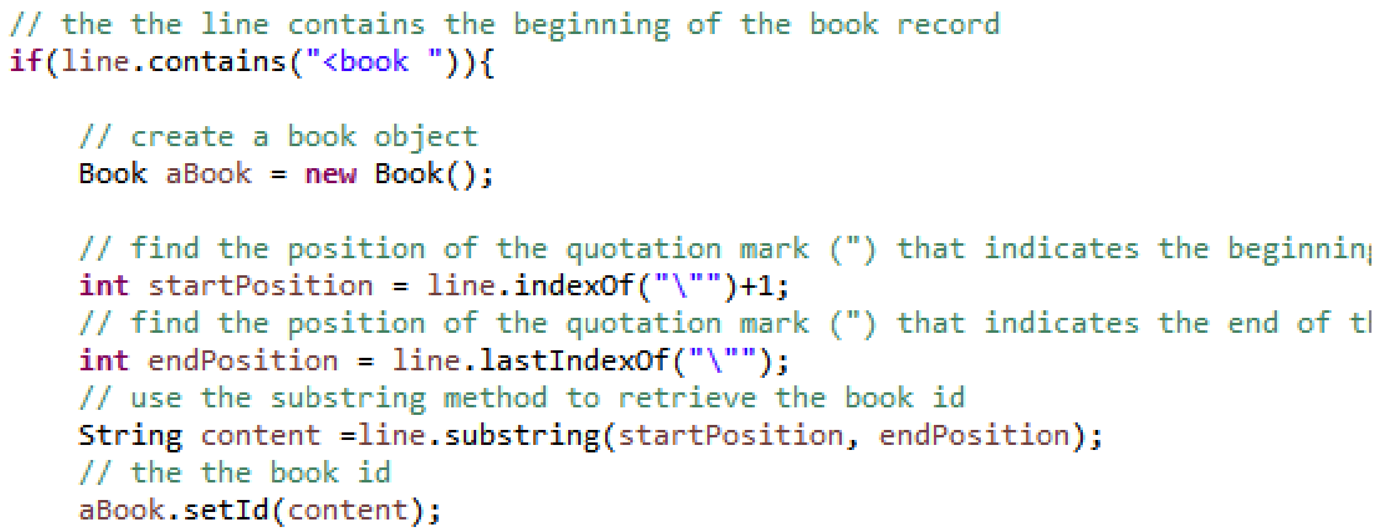
String Processing (2)

- Find the book id



```
<book id="bk101">
```

  - If the line contains **<book**, it represent a book record, with the id information
  - Find the position of the quotation marks
    - indexOf(): returns the first location of a substring or character
    - lastIndexOf(): returns the last location of a substring or character
  - Get the substring between the two quotation marks

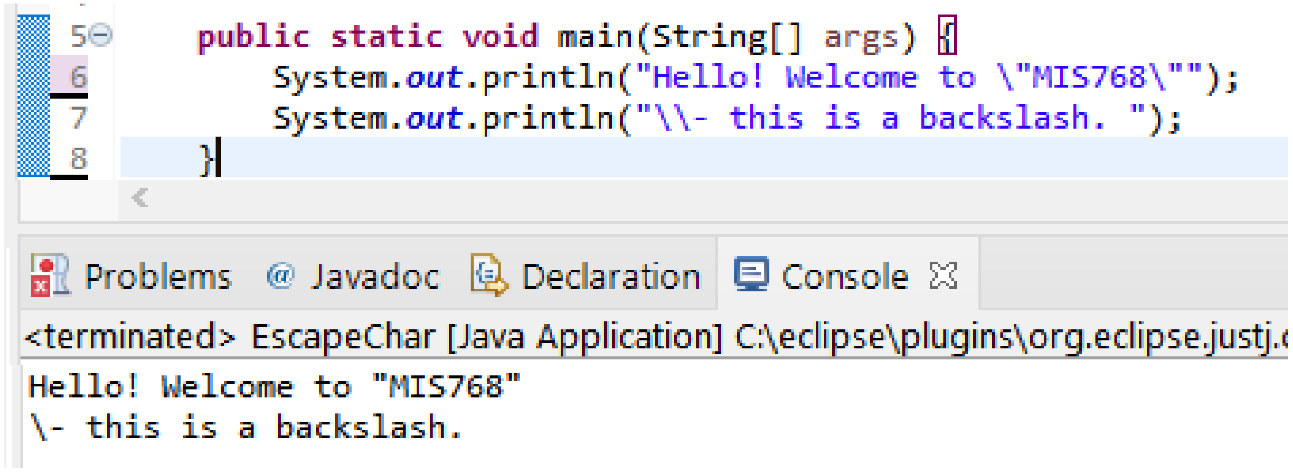


```
// the the line contains the beginning of the book record
if(line.contains("<book ")){

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

    // find the position of the quotation mark (") that indicates the beginnin
    int startPosition = line.indexOf("\"")+1;
    // find the position of the quotation mark (") that indicates the end of t
    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);
```

Escape Character (\)

- Escape character consists of a backslash (\) followed by the character you want to add to the string

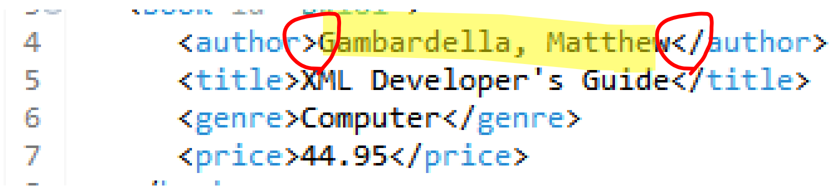
Escape Character	Prints as
\n	Newline (line break)
\t	Tab
\"	Double quote
\\	Backslash



```
    public static void main(String[] args) {
        System.out.println("Hello! Welcome to \"MIS768\"");
        System.out.println("\\- this is a backslash. ");
    }
```

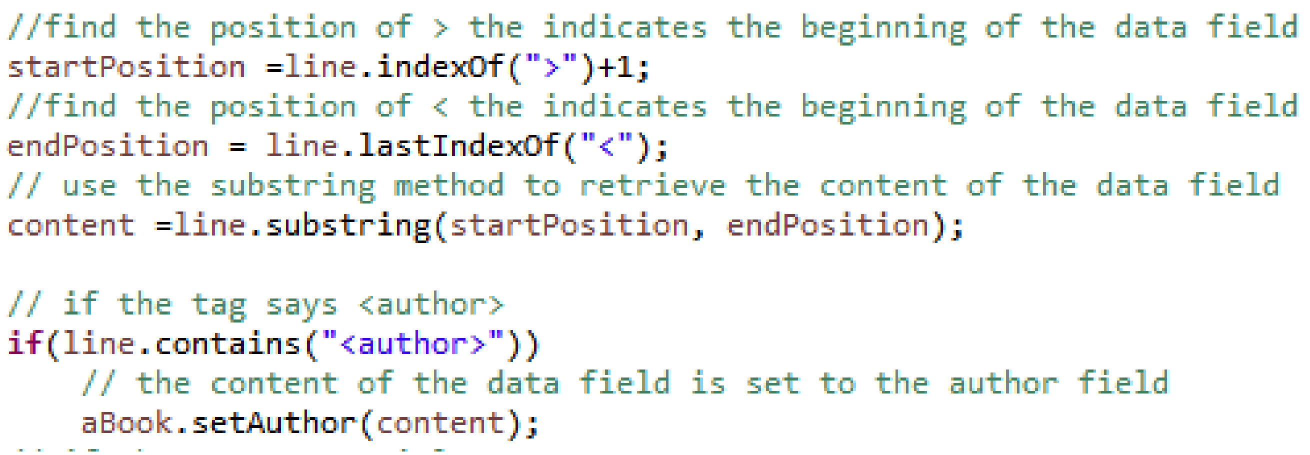
```
<terminated> EscapeChar [Java Application] C:\eclipse\plugins\org.eclipse.justj.
Hello! Welcome to "MIS768"
\- this is a backslash.
```

String Processing (3)



```
<author>Gambardella, Matthew</author>
<title>XML Developer's Guide</title>
<genre>Computer</genre>
<price>44.95</price>
```

- Find the other fields
  - The tags **<>** specify the field name. The values are between the tags
  - Find the position of the first **>** and the last **<**
  - Get the substring between **>** and **<**
  - If the tag is <author>, the values between **>** and **<** is the value for the author field



```
//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);

// if the tag says <author>
if(line.contains("<author>"))
    // the content of the data field is set to the author field
    aBook.setAuthor(content);
```

Enumerated Types (1)

- Known as an **`enum`**, requires declaration and definition like a class
- Syntax:
  `enum` ***`typeName`*** **`{`** ***one or more enum constants*** **`}`**
- Example
  - Definition:

```
enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
              FRIDAY, SATURDAY }
```

  - Declaration:

```
Day workDay; // creates a Day enum
```

  - Assignment:

```
Day workDay = Day.WEDNESDAY;
```

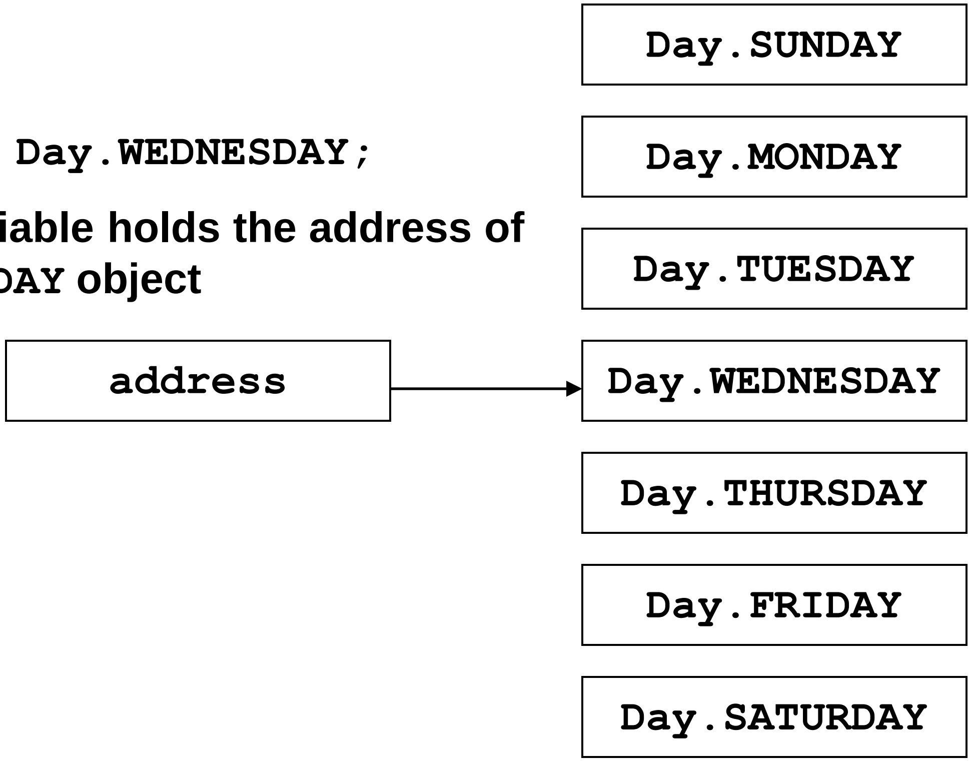
Enumerated Types (2)

- An **`enum`** is a specialized class

```
Day workDay = Day.WEDNESDAY;
```

The `workDay` variable holds the address of the `Day.WEDNESDAY` object

Each are objects of type **Day**, a specialized class



Enumerated Types (3)

- Methods
  - **`toString:`** Returns name of calling constant
  - **`ordinal`**
    - Returns the zero-based position of the constant in the **`enum`**.
    - For example the ordinal for **`Day.THURSDAY`** is 4
  - **`equals`**: Returns true if the argument is equal to the calling **`enum`** constant
  - **`compareTo`**
    - Returns a negative integer if the calling constant's ordinal is less than the argument's ordinal
    - Returns, a positive integer if the calling constant's ordinal is greater than the argument's ordinal
    - Returns zero if the calling constant's ordinal == the argument's ordinal.

Question

- What is the output of the following code?

```
enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
           THURSDAY, FRIDAY, SATURDAY }

public static void main(String[] args)   {
    Day workDay = Day.WEDNESDAY;

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

Answer

WEDNESDAY

Question (2)

- What is the output of the following code?

```
enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
           THURSDAY, FRIDAY, SATURDAY }

public static void main(String[] args)   {
    System.out.println("The ordinal value for " +
                       Day.SUNDAY + " is " +
                       Day.SUNDAY.ordinal());}
    System.out.println("The ordinal value for " +
                       Day.WEDNESDAY + " is " +
                       Day.WEDNESDAYS.ordinal());
}
```

UNLV

Answer (2)

The ordinal value for SUNDAY is 0

The ordinal value for WEDNESDAY is 3

Question (3)

- What is the output of the following code?

```
enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
           THURSDAY, FRIDAY, SATURDAY }

public static void main(String[] args)   {
    if (Day.FRIDAY.compareTo(Day.MONDAY) > 0)
      System.out.println(Day.FRIDAY +
                           " is greater than " +
                           Day.MONDAY);
    else
      System.out.println(Day.FRIDAY +
                           " is NOT greater than " +
                           Day.MONDAY);
         }
}
```

Answer (3)

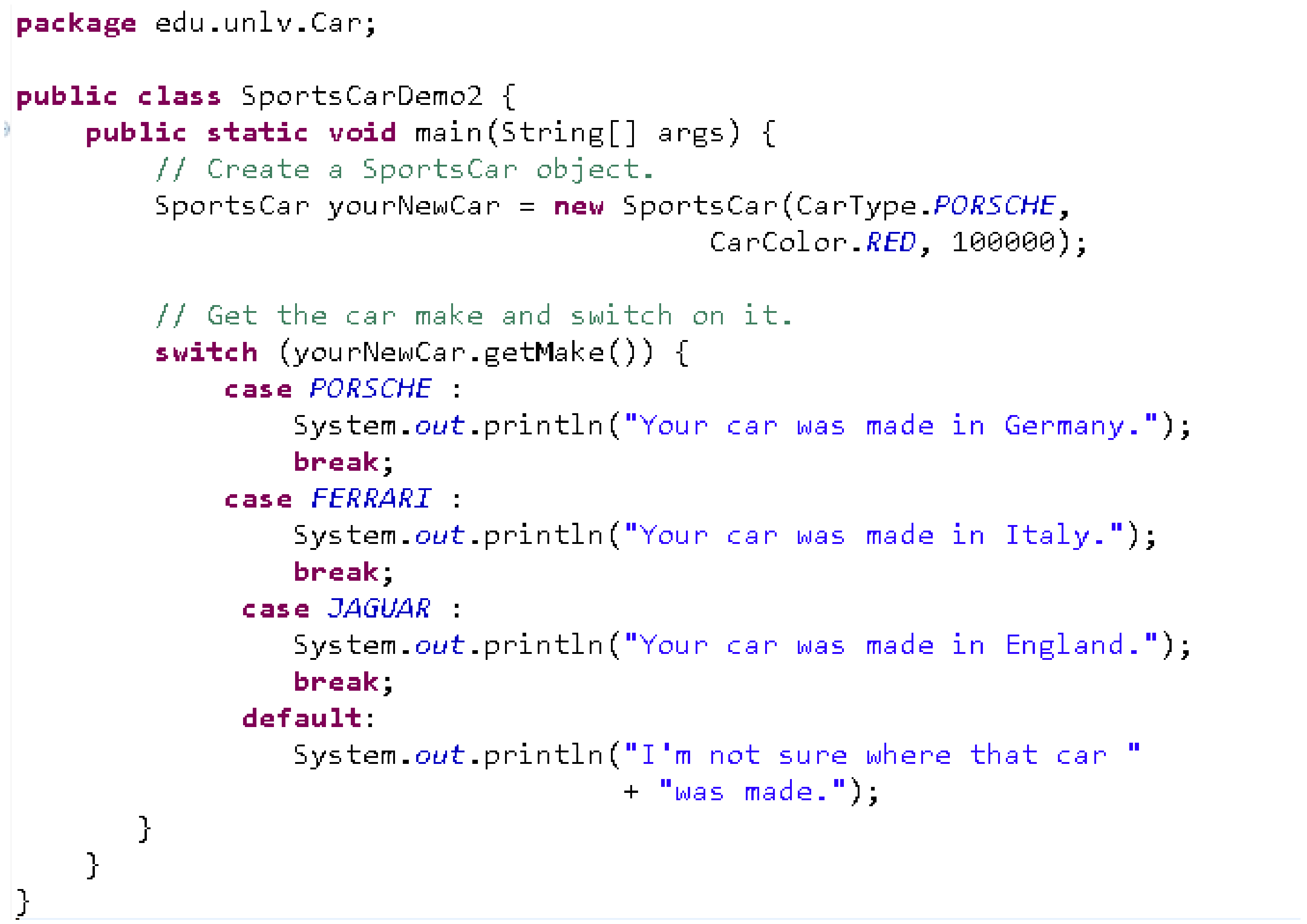
- FRIDAY is greater than MONDAY

Lab (2)

- CarType.java
- CarColor
- SportsCar.java
- SportsCarDemo.java
  - This example illustrates the usage of enum in a program.

Enumerated Types (4)

- Java allows you to test an **`enum`** constant with a **`switch`** statement.



```java
package edu.unlv.Car;

public class SportsCarDemo2 {
    public static void main(String[] args) {
        // Create a SportsCar object.
        SportsCar yourNewCar = new SportsCar(CarType.PORSCHE,
                                        CarColor.RED, 100000);

        // Get the car make and switch on it.
        switch (yourNewCar.getMake()) {
            case PORSCHE :
                System.out.println("Your car was made in Germany.");
                break;
            case FERRARI :
                System.out.println("Your car was made in Italy.");
                break;
             case JAGUAR :
                System.out.println("Your car was made in England.");
                break;
             default:
                System.out.println("I'm not sure where that car "
                                   + "was made.");

        }
    }
}
```

enum with Associated Values

- enum type can be defined with certain value associated with each constant

```
public enum AutoJob {
    OIL_CHANGE (26.00),
    LUBE_JOB (18.00),
    RADIATOR_FLUSH(30.00),
    TRANSMISSION_FLUSH(80.00),
    INSPECTION(15.00),
    MUFFLER_REPLACEMENT(100.00),
    TIRE_ROTATION(20.00);

    private double price;

    AutoJob (double aPrice){
        this.price = aPrice;
    }
    void setPrice (double aPrice){
        this.price = aPrice;
    }
    double getPrice(){
        return this.price;
    }

}
```

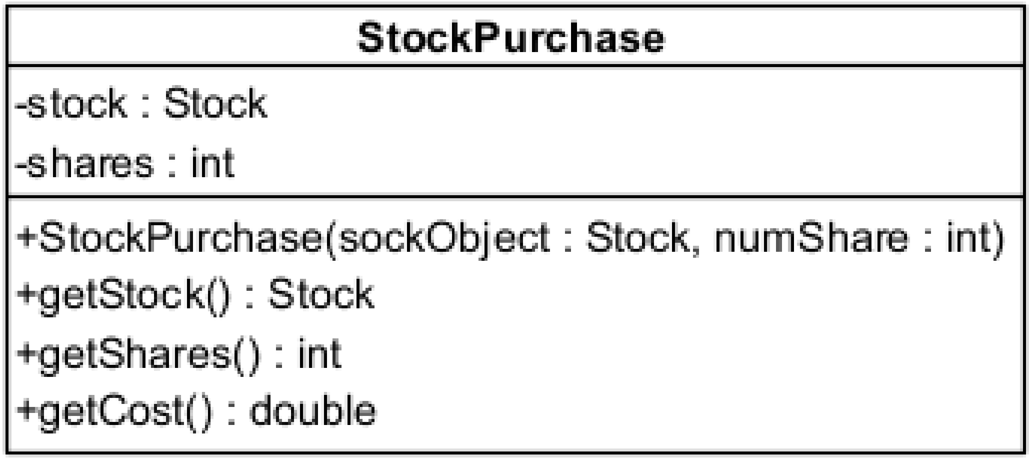
Class Collaboration

- Collaboration – two classes interact with each other
- If an object is to collaborate with another object, it must know something about the second object's methods and how to call them

Class Collaboration: Example (1)

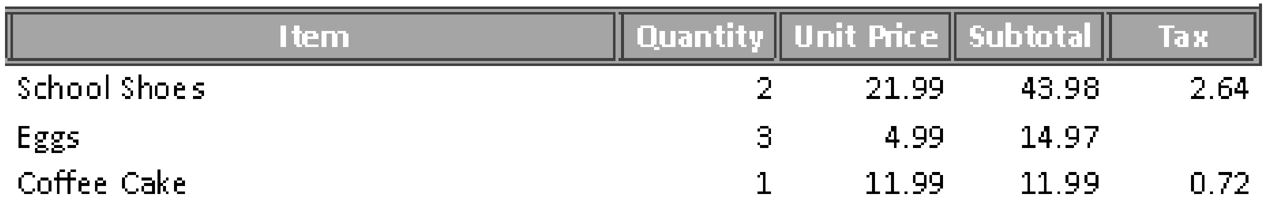
- If we design a class **`StockPurchase`** that collaborates with the **`Stock`** class (previously defined) to simulate the purchase of a stock.
  - The **`StockPurchase`** class is responsible for calculating the cost of the stock purchase.
  - It uses **`Stock`** class’s constructor and the **`Copy`** method to makes a copy of the **`Stock`** object.
  - It must know how to call the **`Stock`** class’s **`getSharePrice`** method to get the price per share of the stock.

Lab (3)



- StockPurchase.java
  - getCost() method can be defined to include the transaction fee or other fees.
- StockTrader.java
  - Application class to interact with the user

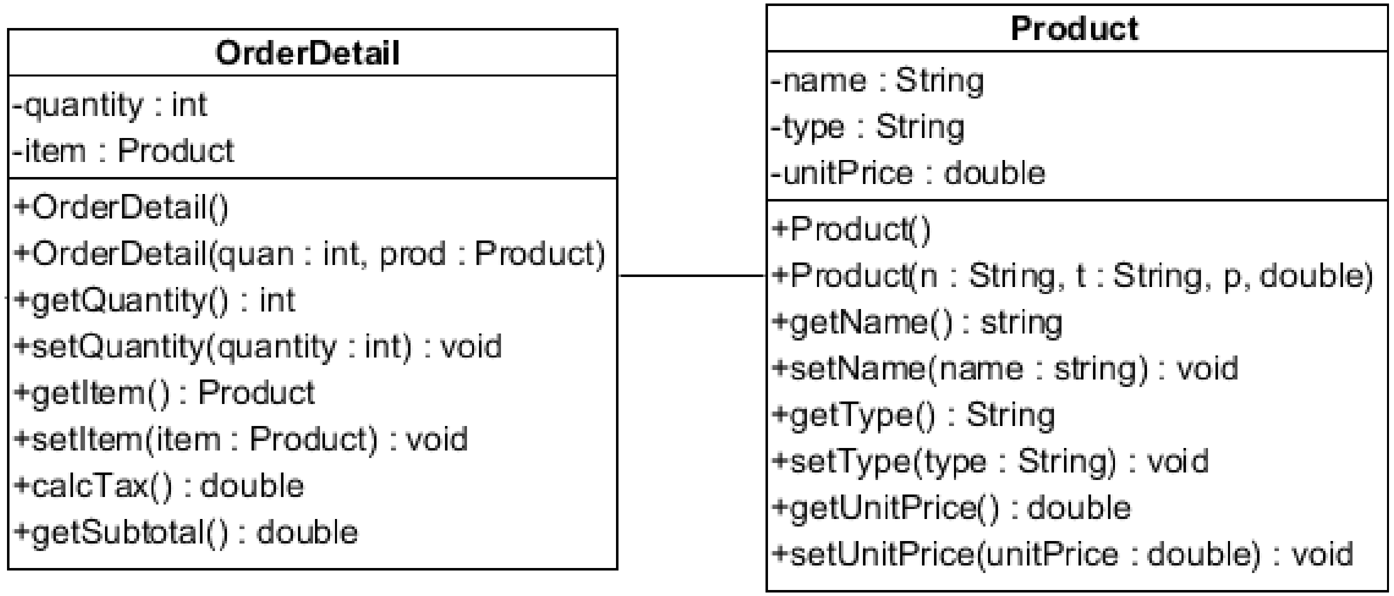
Class Collaboration: Example (2)



Item	Quantity	Unit Price	Subtotal	Tax
School Shoes	2	21.99	43.98	2.64
Eggs	3	4.99	14.97	
Coffee Cake	1	11.99	11.99	0.72

- A line of order detail would include a product, and also the quantity
- Tax is calculated based on the type of the product
- Subtotal is also calculated based quantity and the unit price of the item

Class Collaboration: Example (3)

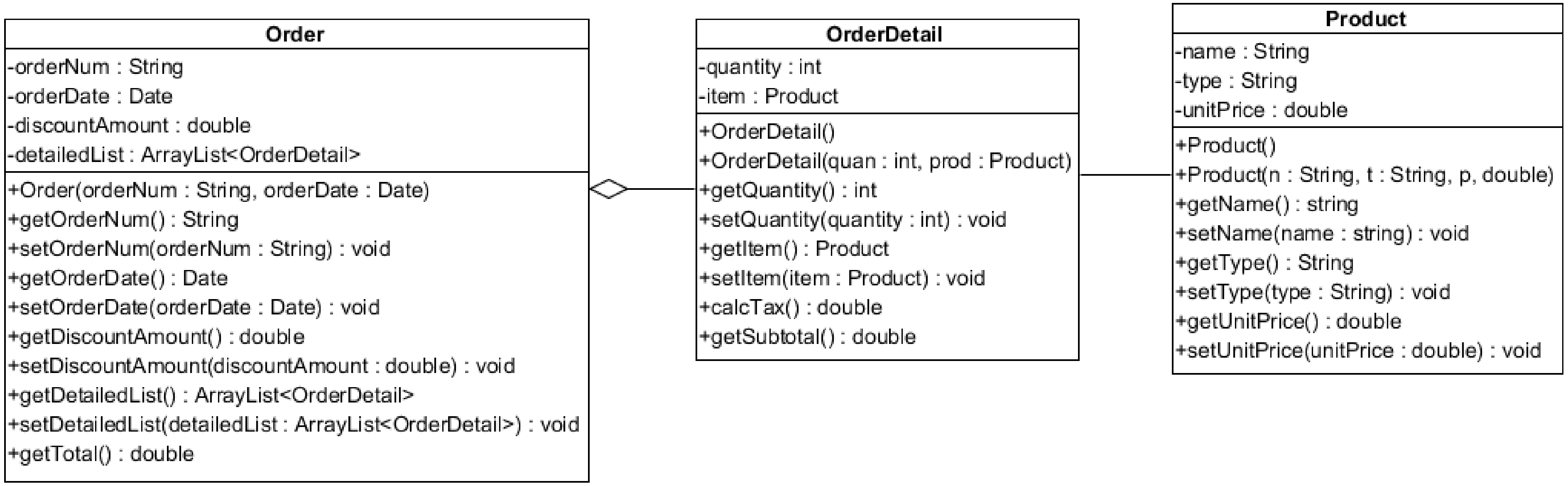


- A line of order detail would include a product, and also the quantity
- Tax is calculated based on the type of the product
- Subtotal is also calculated based quantity and the unit price of the item

Lab (4)

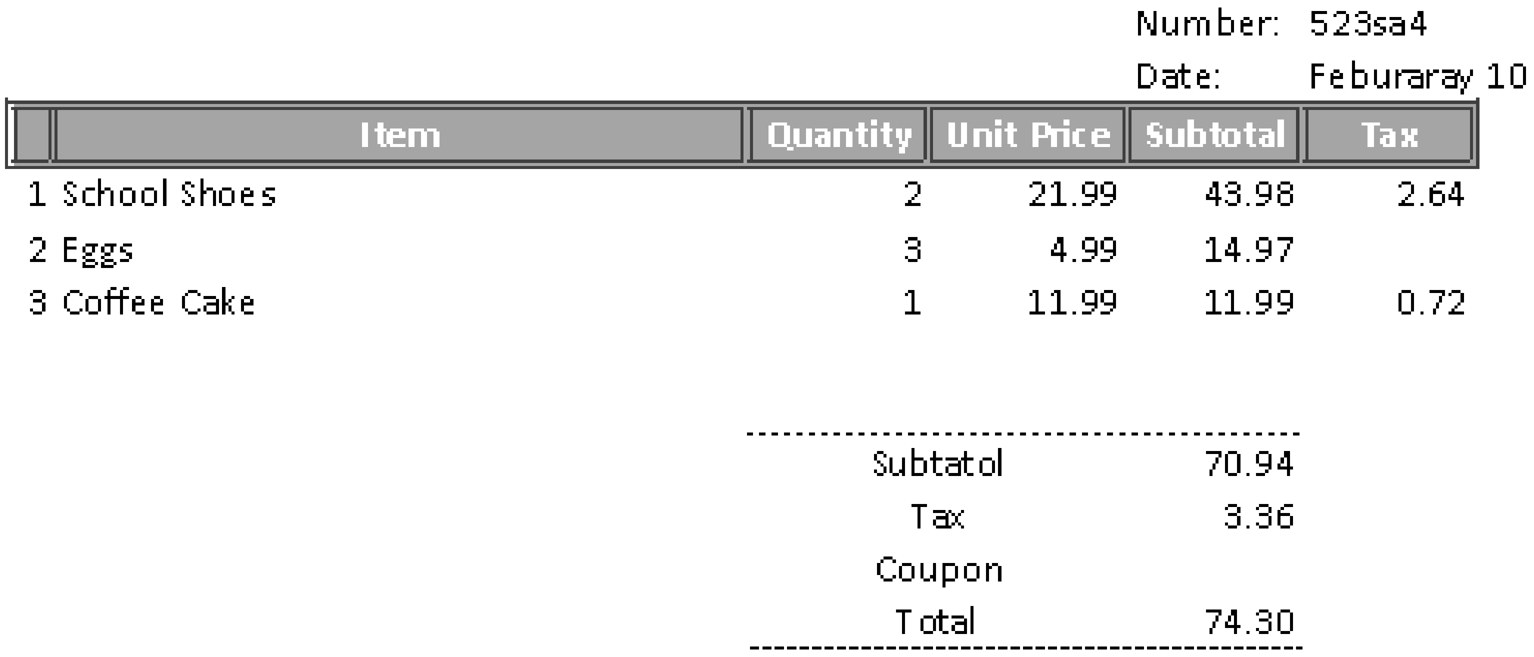
- Product.java
- OrderDetail.java
- OrderDetailDemo.java
  - An application class to test the above two classes

Aggregation of Objects


Order
-orderNum : String
-orderDate : Date
-discountAmount : double
-detailedList : ArrayList<OrderDetail>
+Order(orderNum : String, orderDate : Date)
+getOrderNum() : String
+setOrderNum(orderNum : String) : void
+getOrderDate() : Date
+setOrderDate(orderDate : Date) : void
+getDiscountAmount() : double
+setDiscountAmount(discountAmount : double) : void
+getDetailedList() : ArrayList<OrderDetail>
+setDetailedList(detailedList : ArrayList<OrderDetail>) : void
+getTotal() : double
OrderDetail
-quantity : int
-item : Product
+OrderDetail()
+OrderDetail(quan : int, prod : Product)
+getQuantity() : int
+setQuantity(quantity : int) : void
+getItem() : Product
+setItem(item : Product) : void
+calcTax() : double
+getSubtotal() : double
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

- The field of a Class can be an array or an **`ArrayList`**
- The aggregation of order details is an order.
- Total is determined by the subtotal and the discount/coupon amount entered.

Example: Sales Receipt (1)



Number:	523sa4
Date:	Feburaray 10

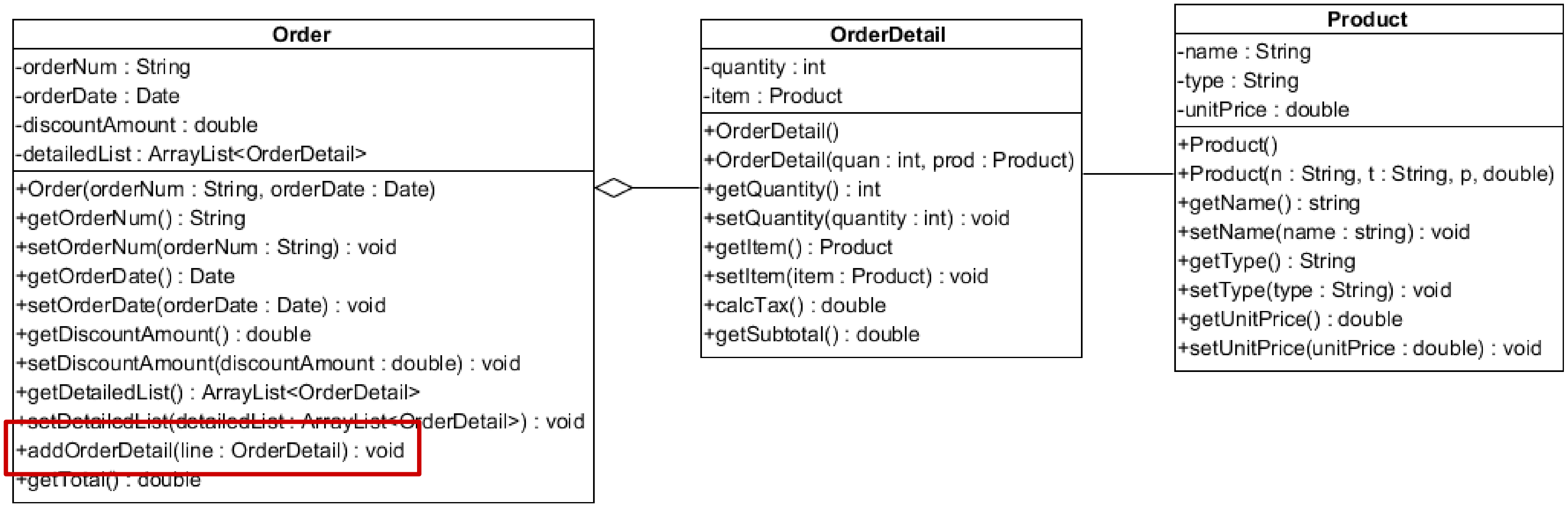
	Item	Quantity	Unit Price	Subtotal	Tax
1	School Shoes	2	21.99	43.98	2.64
2	Eggs	3	4.99	14.97	
3	Coffee Cake	1	11.99	11.99	0.72

Subtatol	70.94
Tax	3.36
Coupon	
Total	74.30

Example: Sales Receipt (2)

- In the **`Order`** Class, a field is a ArrayList of **`OrderDetail`**
  - Number of **`OrderDetail`** objects is not fixed
  - Can use a loop to process all **`OrderDetail`** objects

Example: Sales Receipt (3)



```
Order
-orderNum : String
-orderDate : Date
-discountAmount : double
-detailedList : ArrayList<OrderDetail>
+Order(orderNum : String, orderDate : Date)
+getOrderNum() : String
+setOrderNum(orderNum : String) : void
+getOrderDate() : Date
+setOrderDate(orderDate : Date) : void
+getDiscountAmount() : double
+setDiscountAmount(discountAmount : double) : void
+getDetailedList() : ArrayList<OrderDetail>
+setDetailedList(detailedList : ArrayList<OrderDetail>) : void
+addOrderDetail(line : OrderDetail) : void
+getTotal() : double
```

```
OrderDetail
-quantity : int
-item : Product
+OrderDetail()
+OrderDetail(quan : int, prod : Product)
+getQuantity() : int
+setQuantity(quantity : int) : void
+getItem() : Product
+setItem(item : Product) : void
+calcTax() : double
+getSubtotal() : double
```

```
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
```

- In addition to setting the list all at once, an additional method can be provided to add one element at a time

Lab (5)

- Order.java
  - Add two methods to complete the implementation
- OrderDemo.java
  - It simulates the check-out process