CS1002 W11 Practical Student ID: 150012773 Tutor: Shyam Reyal 23 Nov 2015

Overview

The week 11 practical required we model a given specification using a UML class diagram and then write the corresponding java code in intellJ. To efficiently complete the practical, we were asked to include relevant associations, with navigability and multiplicity information. We were also asked to use inheritance where appropriate and ensure that the classes we created are properly encapsulated, contain appropriate setters, getters, and constructors, and provide methods to implement any required functionality. Finally it was required we write and run some JUnit tests to demonstrate that the main features work correctly.

Design

It should be noted that all comments and the jUnit tests for the main class can be found in the extension. I have also added an extra constructor to the 'Book' class in the extension which sets the book type and finally I have changed some of the printing formats in the extension file. I have also included the java doc in the zip folder uploaded to MMS.

The basic design emulated an online bookshop. The book shop has two types of items for sale: books and stationery. All items have a unique item code, name, quantity available and price. In addition, books have one or more authors, a publisher, ISBN, genre and publication date. Books can be in 3 possible formats: paperback, hardback or electronic. Stationary items have a type (diary, notebook, etc). All items, except electronic ones, incur a flat delivery charge of £1.00 each. An author has a name, address, one or more genres he / she writes in, and a contact number for his / her agent. Customers can add one or more items from the shop to a personal shopping basket. The delivery charge is waived for purchases worth at least £20. Each customer has a name, delivery address and email address."

In this practical I have added a variety of extensions. The extensions include: allowing a shop to have a one or more employees whom are able to process orders made by the customers. Each employee has an id, name, address, annual salary and contact telephone number. Each order (contents of a basket) is dealt with by one employee. The next extension allows the system to print all employees whose total sales are above a set amount, and the ability to print all customers who have bought a particular item. For the next extension I decided to incorporate a class 'Account' which emulates a basic bank. This allows customers to pay for their order. Each customer starts off with a predefined balance in their account, when orders are made their balance is decreased, moreover when an order is cancelled they are reimbursed and their account is debited the amount of the order. Finally, I have included the notion of commission and enabled my program to decide whether each employee is entitled to a bonus and if so how much. Throughout my program I have also used techniques such as try and catch statement and polymorphism.

To allow unique ID's for each individual object I have used static variables which increment when each new object is made and thus allows unique IDs for all objects of the given class. To start with I made two super classes 'Item' and 'People'. The item class contains the attributes that are common in the in the subclasses, 'Book', 'ElectronicBook' and 'Stationary' for example unique item code. In this class there is are various getters and setters, a constructor which also sets the the unique item code for each new object and an overridden, toString statement. The next superclass, 'Person' contained the name attribute which is common to the classes, 'Author', 'Customer' and 'Employee'. This class has a constructor, a getter and setter for the name attribute and an overridden, toString statement. Using inheritance in the is way meant I was able not only save space but my code easier to manage, change and maintain.

Through my program where I have used inheritance and then overridden a method, which I have often done with the constructor methods, I have used the super keyword in java to invoke the overridden method.

I used array lists through out my program because they are easier to work compared to a regular array, and despite slower access speed they do not require a predefined fixed size. When manipulating any of the array lists in my program I have opted to use the try and catch exception handlers so that the program can attempt to run a piece of code, which might produce and error and if in the event that it does the program can handle the error accordingly, rather than crashing.

When searching for an item from an array list, the program preforms a liner search on the array in question, until the item is found and which it is removed or if the search reaches the end of the array. The reason I chose to use a linier search is because it is very simple to code and despite not being as quickest search, the small data set meant speed was not an issue.

I have also added multiple getter and setter method to to all my which allows me to make attributes in the classes private or protected and thus enforcing encapsulation as well as allowing me to manipulate the attributes of each object and return information on objects.

The EnumClass was the next class I decided to create. This class contains two enum types, genre and type. I did this so I could represent a fixed set of constants that I could then use throughout my program.

I then created the 'Author' class, which extends the 'Person' class. This class contains an array list of class type 'Book' and array list of type 'Genre'. These two array lists allowed me store the books each author has written the genre(s) the author writes in.

Next I created the 'Customer' class which extends the 'Person;' class, this class contains a constructor and various attributes including an array list of 'Items' which acts as the customer's basket and a Boolean attribute, 'orderComplete' which allows the customers basket to be processed. The core methods in the 'Customer' class include: 'removeItem', which removes an item from the customer's basket by preforming a linier search on the array list, 'basketTotal' which cycles through each item in the customer basket whilst incrementing a counting variable and calculates the total of the basket, this method then calculates the cost of postage, which is £1 for physical item or free if the total is over £20. Another method allows a customer to view all the items in their basket.

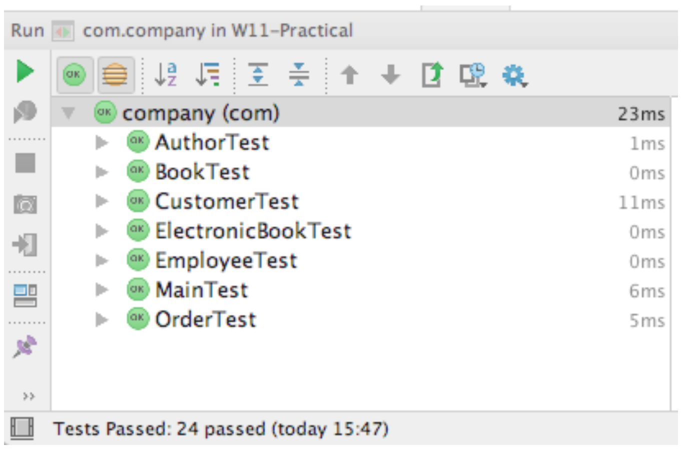
After that I created the 'Book' class which extended the 'Item' class, this class contains: a constructor and various attributes including a list array of type 'Author' an 'EnumClass.genre' attribute to store the genre of the book and a String attribute 'type' which is used to store the book type; paper back or hard back. The core methods in the 'Book' class include: 'addAuthor' which adds an author to list of authors for a given book and 'removeAuthor' which removes an author from the authors array by preforming a linier search on the array list. I decided then to make the 'ElectronicBook' class extend the book class as they are very similar but an electronic book can have a memory size, so therefore I added this attribute to the 'ElectronicBook' class and that is the only difference. The final class to extend the 'Item' class is the stationary class. Similary to that of the book class the 'Stationary' class contains an enum type attribute called 'stationaryType'

The following classes I created where for the extension activities. The first class I created, for the first extention activity, was the 'Employee' class, which extends the 'Person' class. This class contains a constructor and various attributes including a static array list of 'Orders' which stores all the orders processed by all the employees, 'totalSales' which stores the total of all the order totals, processed by that employee, 'bounus' which is used to store bonus earned by each employee, this is worked out by using the 'totalSales' attribute and 'currentOrder' which stores the order that an employee is currently processing. The core methods in the employee class are: 'deleteOrder' which is used if an order is deleted by the customer, or employee. The method updates the stock of the items in the order and the employee's total sales is also amended, finally, the method credits the customers account. The method takes the orderID of the order and preforms liner search to find the order. The next method is the 'processOrder' method, this method adds the order to the orders array and updates an employee's total sales.

An orders class was also required to complete the first extension, so I created the 'Orders' class next. This class contains a constructor and various attributes including: a array list of 'Items which stores all the items in a given order, 'customer' which is used to store the customer whom placed the given order, 'enployee' which is used to store the employee processing the given order, 'orderDate' which stores the date the order was made, 'orderTotal; that stores to total cost of the order, including postage and finally 'noOfItemsOrdered' that is used to keep track of how many items where actually in a given order. The core method in the order class is the 'addOrder' method. When an employee processes an order this method is executed the order total is recorded and the number of items is also recorded. Every item in the current order is added to an array and the quantity of each item in the order is updated accordingly, next the customers account is the charged the total of the order. Finally, the customer's basket is reset and the order complete attribute is set to false.

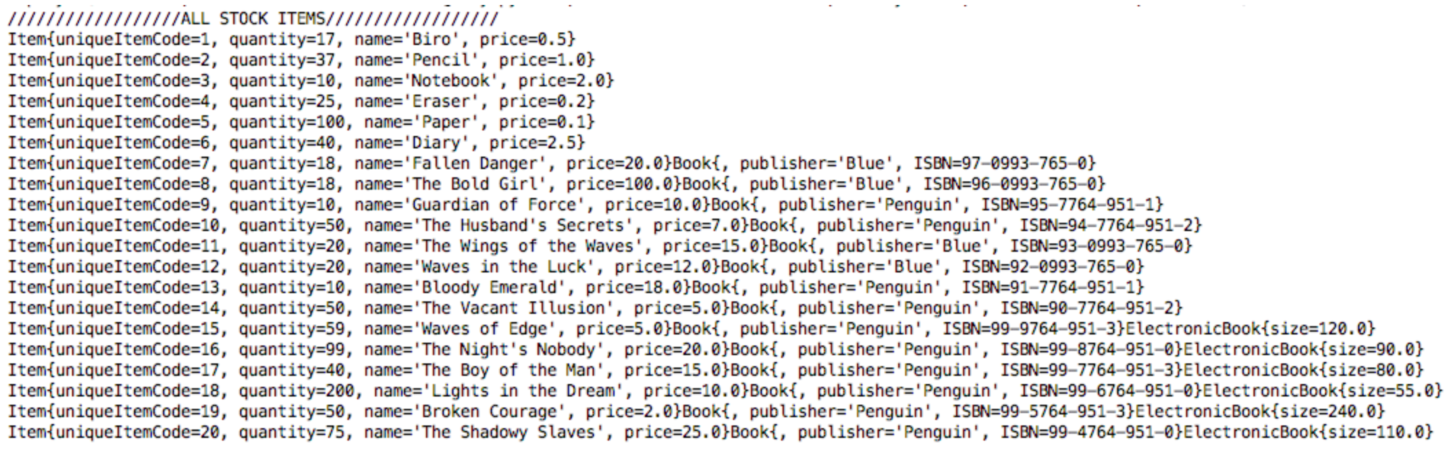
Testing

To test my program works as expected and that the methods function as intended I have written and run various JUnit tests that demonstrate the main features work correctly. So that the tests cover a suitable amount of code there is at least one test case for each requirement in the practical specification I have also tested all the complex methods in my program extensively ewith a verity of different test data ensuring all the outcomes are as expected. Because some features are not testable by JUnit (e.g. printed outputs to System.out). I have created a main class with a main method. I have then processed to create instances of all classes, the objects are linked appropriately by calling methods or passing objects to constructors. The details of the objects are then printed to the command line and compared to expected results. There are no instances where the user needs to input data into the program, I will not have to do to any verification and validation tests. Moreover, because I have also incorporated many try and catch error statements, erroneous and extreme testing will be automatically covered.



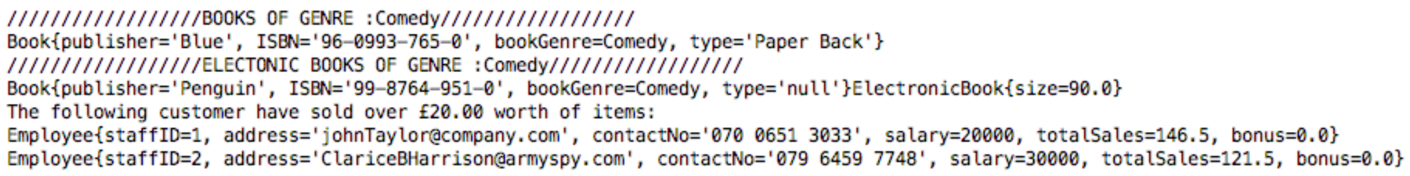
```
Run  com.company in W11-Practical
company (com)          23ms
  AuthorTest            1ms
  BookTest              0ms
  CustomerTest         11ms
  ElectronicBookTest    0ms
  EmployeeTest          0ms
  MainTest              6ms
  OrderTest             5ms
Tests Passed: 24 passed (today 15:47)
```

Here you can see all 24 tests from all the classes passed



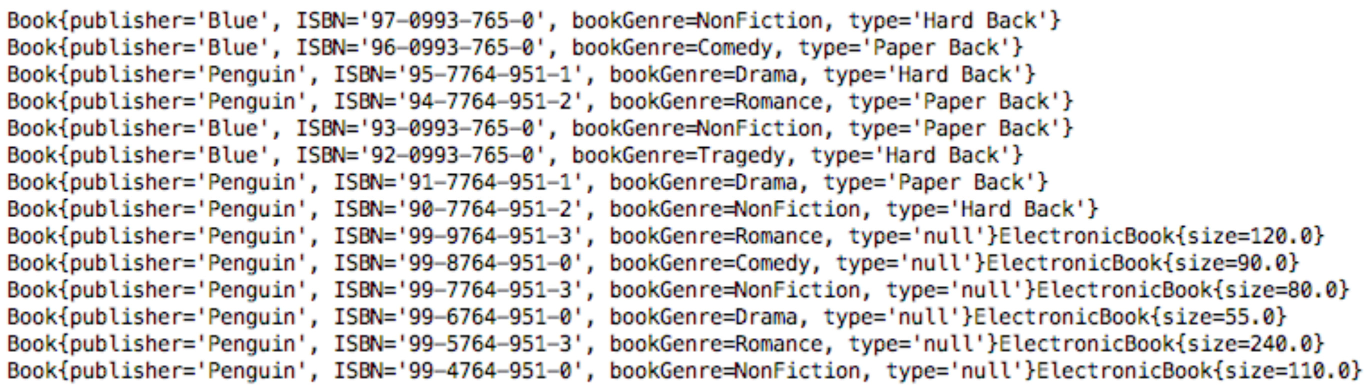
```
//////////////////ALL STOCK ITEMS//////////////////
Item{uniqueItemCode=1, quantity=17, name='Biro', price=0.5}
Item{uniqueItemCode=2, quantity=37, name='Pencil', price=1.0}
Item{uniqueItemCode=3, quantity=10, name='Notebook', price=2.0}
Item{uniqueItemCode=4, quantity=25, name='Eraser', price=0.2}
Item{uniqueItemCode=5, quantity=100, name='Paper', price=0.1}
Item{uniqueItemCode=6, quantity=40, name='Diary', price=2.5}
Item{uniqueItemCode=7, quantity=18, name='Fallen Danger', price=20.0}Book{, publisher='Blue', ISBN=97-0993-765-0}
Item{uniqueItemCode=8, quantity=18, name='The Bold Girl', price=100.0}Book{, publisher='Blue', ISBN=96-0993-765-0}
Item{uniqueItemCode=9, quantity=10, name='Guardian of Force', price=10.0}Book{, publisher='Penguin', ISBN=95-7764-951-1}
Item{uniqueItemCode=10, quantity=50, name='The Husband's Secrets', price=7.0}Book{, publisher='Penguin', ISBN=94-7764-951-2}
Item{uniqueItemCode=11, quantity=20, name='The Wings of the Waves', price=15.0}Book{, publisher='Blue', ISBN=93-0993-765-0}
Item{uniqueItemCode=12, quantity=20, name='Waves in the Luck', price=12.0}Book{, publisher='Blue', ISBN=92-0993-765-0}
Item{uniqueItemCode=13, quantity=10, name='Bloody Emerald', price=18.0}Book{, publisher='Penguin', ISBN=91-7764-951-1}
Item{uniqueItemCode=14, quantity=50, name='The Vacant Illusion', price=5.0}Book{, publisher='Penguin', ISBN=90-7764-951-2}
Item{uniqueItemCode=15, quantity=59, name='Waves of Edge', price=5.0}Book{, publisher='Penguin', ISBN=99-9764-951-3}ElectronicBook{size=120.0}
Item{uniqueItemCode=16, quantity=99, name='The Night's Nobody', price=20.0}Book{, publisher='Penguin', ISBN=99-8764-951-0}ElectronicBook{size=90.0}
Item{uniqueItemCode=17, quantity=40, name='The Boy of the Man', price=15.0}Book{, publisher='Penguin', ISBN=99-7764-951-3}ElectronicBook{size=80.0}
Item{uniqueItemCode=18, quantity=200, name='Lights in the Dream', price=10.0}Book{, publisher='Penguin', ISBN=99-6764-951-0}ElectronicBook{size=55.0}
Item{uniqueItemCode=19, quantity=50, name='Broken Courage', price=2.0}Book{, publisher='Penguin', ISBN=99-5764-951-3}ElectronicBook{size=240.0}
Item{uniqueItemCode=20, quantity=75, name='The Shadowy Slaves', price=25.0}Book{, publisher='Penguin', ISBN=99-4764-951-0}ElectronicBook{size=110.0}
```

Here you can see that all the items in stock and their corresponding details have been successfully printed to the command line.



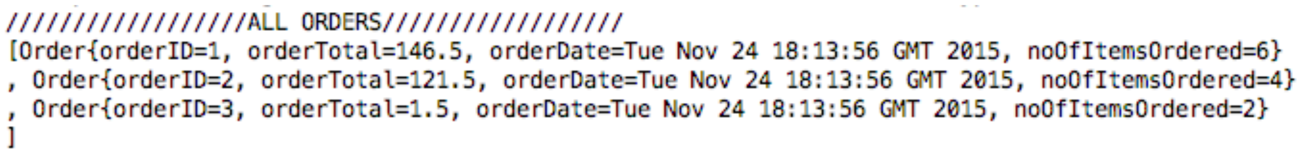
```
//////////////////BOOKS OF GENRE :Comedy//////////////////
Book{publisher='Blue', ISBN='96-0993-765-0', bookGenre=Comedy, type='Paper Back'}
//////////////////ELECTONIC BOOKS OF GENRE :Comedy//////////////////
Book{publisher='Penguin', ISBN='99-8764-951-0', bookGenre=Comedy, type='null'}ElectronicBook{size=90.0}
The following customer have sold over £20.00 worth of items:
Employee{staffID=1, address='johnTaylor@company.com', contactNo='070 0651 3033', salary=20000, totalSales=146.5, bonus=0.0}
Employee{staffID=2, address='ClariceBHarrison@armyspy.com', contactNo='079 6459 7748', salary=30000, totalSales=121.5, bonus=0.0}
```

Here you can see that all the books created and their corresponding details have been successfully printed to the command line.



```
Book{publisher='Blue', ISBN='97-0993-765-0', bookGenre=NonFiction, type='Hard Back'}
Book{publisher='Blue', ISBN='96-0993-765-0', bookGenre=Comedy, type='Paper Back'}
Book{publisher='Penguin', ISBN='95-7764-951-1', bookGenre=Drama, type='Hard Back'}
Book{publisher='Penguin', ISBN='94-7764-951-2', bookGenre=Romance, type='Paper Back'}
Book{publisher='Blue', ISBN='93-0993-765-0', bookGenre=NonFiction, type='Paper Back'}
Book{publisher='Blue', ISBN='92-0993-765-0', bookGenre=Tragedy, type='Hard Back'}
Book{publisher='Penguin', ISBN='91-7764-951-1', bookGenre=Drama, type='Paper Back'}
Book{publisher='Penguin', ISBN='90-7764-951-2', bookGenre=NonFiction, type='Hard Back'}
Book{publisher='Penguin', ISBN='99-9764-951-3', bookGenre=Romance, type='null'}ElectronicBook{size=120.0}
Book{publisher='Penguin', ISBN='99-8764-951-0', bookGenre=Comedy, type='null'}ElectronicBook{size=90.0}
Book{publisher='Penguin', ISBN='99-7764-951-3', bookGenre=NonFiction, type='null'}ElectronicBook{size=80.0}
Book{publisher='Penguin', ISBN='99-6764-951-0', bookGenre=Drama, type='null'}ElectronicBook{size=55.0}
Book{publisher='Penguin', ISBN='99-5764-951-3', bookGenre=Romance, type='null'}ElectronicBook{size=240.0}
Book{publisher='Penguin', ISBN='99-4764-951-0', bookGenre=NonFiction, type='null'}ElectronicBook{size=110.0}
```

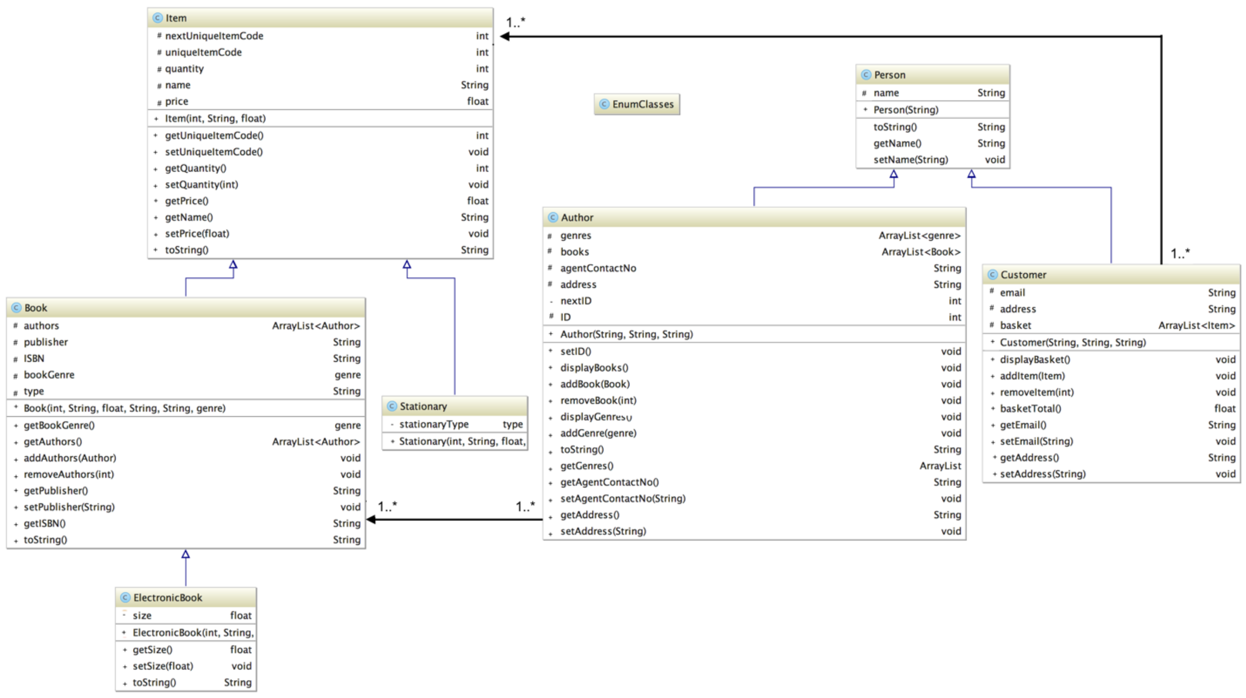
Here you can see that all the books of a certain genre in this case, comedy, created and their corresponding details have been successfully printed to the command line.



```
//////////////////ALL ORDERS//////////////////
[Order{orderID=1, orderTotal=146.5, orderDate=Tue Nov 24 18:13:56 GMT 2015, noOfItemsOrdered=6}
, Order{orderID=2, orderTotal=121.5, orderDate=Tue Nov 24 18:13:56 GMT 2015, noOfItemsOrdered=4}
, Order{orderID=3, orderTotal=1.5, orderDate=Tue Nov 24 18:13:56 GMT 2015, noOfItemsOrdered=2}
]
```

Here you can see that all the orders made and their corresponding details have been successfully printed to the command line.

<u>UML Diagram</u>



Evaluation

The program I designed meet all the initial requirements successfully as well as incorporating various different extensions. After testing all the features of my program, I pleased to say that the program works how I expected it too with all the methods functioning properly the program is also able to with erroneous and extreme tests without crashing in most cases, due to the try and except statements, in the event of an error, a description of the error is printed to the command line. One possible drawback is that the user had manually change the code to add and remove objects and this could cause errors as some objects need to be initialised in certain orders for example you can't create an order before you create an item, customer and employee.

Conclusion

In conclusion I'm satisfied with my solution to the initial and extension activities. My program has all the required plus added functionality which make the program more flexible and user friendly. The extensions caused some difficulty as the complexity of the program greatly increased with many various relationships between classes. One other thing I may change is adding two more classes, 'HardBack' and 'PaperBack'. However, this is not essential and doing so would in no way alter the over all functionality of the code. For this practical I would have liked to create and use some kind of simple interface so the user can input information or further to create a graphical user interface.