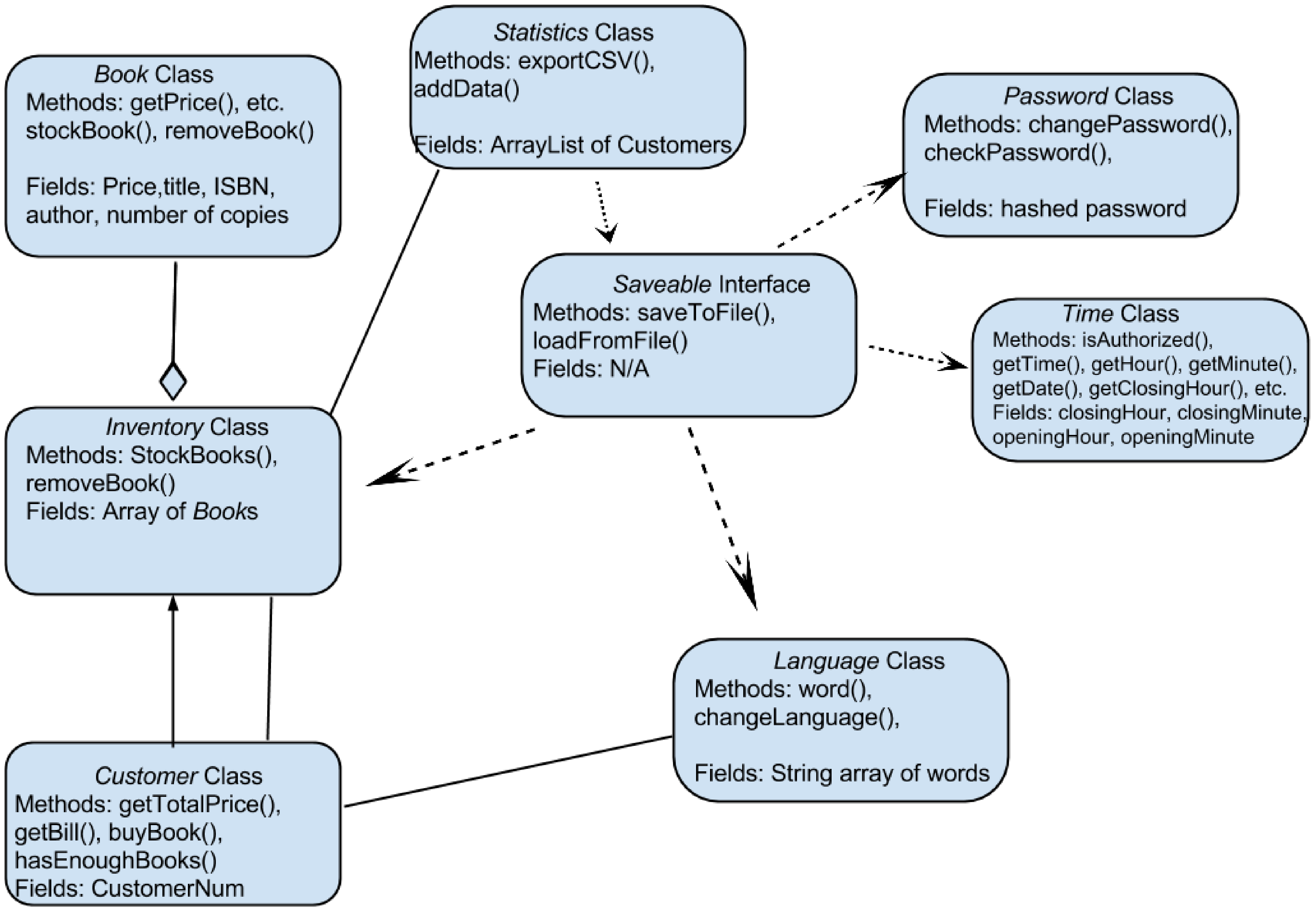
Final Project Description

Vishwesh Anand Ronak Sharma

Our final project consists of 8 different classes and one interface. The classes are: Book, Customer, Inventory, Language, Password, Statistics, and Time. The interface that we made is called the Saveable interface. The classes that implement this interface are customer, inventory, language password, statistics, and time. The book class is class that basically just constructs a book object with a title, author, isbn number, price, and how many copies of that book are available. In the book class we have methods to return each parameter of a book object in separate fields. The getTitle() method returns the title of a constructed book. The getAuthor() method returns the author of the specified book. The stockBook() and removeBook() methods add or subtract from the quantity field of the class. In addition to the book class, we also designed a inventory class. In the inventory class, the methods we have put are addBook(), removeBook(), sort(), createBook(), updateFile(), saveToFile(), and loadFromFile(). The inventory class basically stores all the copies of the books that are available in the store. The addBook() method adds another copy of a book if the inventory “runs” out of the specific book. The removeBook() method removes a book from the inventory array when a person “buys” the same book. The sort() method sorts the array by ISBN number as stated in the project guide. The createBook() “creates a new book in the array. The updateFile() method updates the save file that we save the inventory to at the end of the day. The saveToFile() saves the inventory array to a new file. The loadFromFile() method allows the user to load an array from a file. We also have a Customer class, which stores all of the books that a customer is ordering, due to the fact that it has similar functionality to the Inventory class as they both store an array of books, we have decided to make Customer extend Inventory, and thereby inherit many of the similar methods, the customer class also has a getBill()

method that returns a bill formatted in HTML in any language, the getBill() method also adds the bill onto a file named bills.html, which is a record of every bill created by the program. We also designed and tested a Time class, in order to ensure that the system is not tampered with by hackers when the store is closed. The time class sets the opening and closing hour of the store and gets the current time in reality. A time object has an opening hour, an opening minute, a closing hour, and a closing minute. The methods we have added in the time class are methods that access and mutate each parameter of a time object. There are getTime() and getDate() methods to return string representations of the time and date respectively, and there is an isAuthorized() method that returns true if the store is open, and false otherwise, there are also methods to return the fields and to get the current hour and minute.The parameter for a statistics object in the statistics class has a parameter of an Inventory object since it needs to see the details of the inventory in order to keep sales data. In the statistics class, we have added methods to add data, and also to export the statistics of each day to a spreadsheet (.csv file openable in Microsoft Excel). The password class has methods which allows the user to create a password, change a password, and make a new password, and check if an inputted password is correct. Also, we have made the password hashed when its stored to a file. That will allow the user to store into a file without having any problems. The Language class works by first loading a list of phrases in a specific language into a string array, to retrieve phrases, the getPhrase() method is used with an integer constant as the parameter, the integer constant represents the line number the phrase is on. We premade a Spanish language file and an English one, but they can be edited to support any language that is represented in Unicode. The language class is used by our GUI to make components and also by the customer class to display a bill. Our Inventory class, Time class, Statistics class, Password class, and Language class all need to save and load their data to a file, so this was the perfect time to utilize the interface feature of Java, so we made a Saveable interface, this interface has two methods, saveToFile() and loadFromFile(), to accomplish those actions. For each specific class, we implemented the saving and loading in a different way, by first planning out a file format, and then implementing the format in both the save and load methods. For example, in the inventory class, it saves to INVENTORY.set, and the first line is the amount of items in the inventory, and then the next five lines are the five fields of the first object, then the five fields of the next object and so on. Due to the design of our classes not testing for validity of the file before using them, the program can crash and return null pointer exceptions if any of the files are modified outside of the program, but if they are not tampered with, an error will not appear, and if the file gets deleted, the class starts the file again, so as long as configuration files are not modified outside the program, the program will not crash. For testing all of our classes, we made a tester class for each of our core classes. In each tester class we had code to test each method and each constructor in each of our core classes. In doing so we were able to find a bug in our statistics class which was affecting our output for our statistics class. With that bug in our program, the data that needed to be exported with the .csv was not being exported correctly. When we tried exporting the data Microsoft Excel would open,but the data wouldn't show up. It would just be a blank spreadsheet. After about a several tries we finally looked through our code, and found the error, which was a == in a string instead of a .equals(). We fixed it and the program exported the statistics correctly. After that little setback, we tested all our classes using the tester classes we made. After that was done, we ran the GUI, fixed some errors, and then we were done with the testing process. We are now satisfied with our program. Some of the changes we made from our initial project proposal are that we added two new classes that we did not have at the beginning of our project. Those classes were the statistics class which we added during the second week, and the language class which we added last weekend. We added those two classes to give our code more features that would enhance the program as well as the user’s experience. Some other changes we made were to our core classes that we had at the beginning of the project. We made numerous changes to the core classes because we had to make them so that they could run even with the new languages and statistics features which we added in the middle of the project. The parts that were easy for our group to do were coming up with the design of our project and deciding what additional features we wanted to have in it. A difficulty we had were updating each core class after each time we added a new feature. We were able to overcome that difficulty by first updating the class, then adding the new feature so that it could support it. Another difficulty we had was when we first attempted the inventory editor, we tried to use JLabels and update them from another method in the Main class, the problem was giving the method access to components inside of the main() method, and another problem was organising the JLabels and JButtons and all of those, we fixed the problem by using an array of JButtons, not using JLabels to make it less complicated, and we also moved the update method to the inner class that houses the actionlistener, so we solved the issue. If we had more time to complete this project, I think we could have added more languages so the GUI could have more language options to run. In addition, another feature we could have added if we had the time would be adding a feature which would allow the program to print the bill of a specified customer out using a printer. What we learned from doing this project is that making code for someone to use to do a task take time and patience. Nobody can just make a program really quick. It’s a long process in which you have to make a design, implement the design in the code, test the code, and present it. If we restarted this project, we think we would have thought about our design better. We would have put more thought in our design because we made A LOT of changes during the project to support what we wanted to do and what features we wanted to add.


Book Class
Methods: getPrice(), etc.
stockBook(), removeBook()
Fields: Price,title, ISBN,
author, number of copies
Statistics Class
Methods: exportCSV(),
addData()
Fields: ArrayList of Customers
Password Class
Methods: changePassword(),
checkPassword(),
Fields: hashed password
Saveable Interface
Methods: saveToFile(),
loadFromFile()
Fields: N/A
Time Class
Methods: isAuthorized(),
getTime(), getHour(), getMinute(),
getDate(), getClosingHour(), etc.
Fields: closingHour, closingMinute,
openingHour, openingMinute
Inventory Class
Methods: StockBooks(),
removeBook()
Fields: Array of *Books*
Language Class
Methods: word(),
changeLanguage(),
Fields: String array of words
Customer Class
Methods: getTotalPrice(),
getBill(), buyBook(),
hasEnoughBooks()
Fields: CustomerNum