OnlineStore:
This is the base class for all this entire project, as such it only have the main and the properties of the Store itself, a list of Users using it, a list of Items that are being sold or already have been, the packages available for the items, as well as the total cost and profits of the store. As in this project we were asked to add all the functionality to manage an Online Store we have not developed any kind of GUI (Graphical User Interface) or menu to use it as our efforts were centered around the functionality.

Item:
This class has all the information that needs to be used, including the Seller of the item itself. We have the method assignbestPackage(List<Package> p) to get the most suitable package of all the packages available, and if there is none it prints to screen that there is no package available and we should introduce a package that suits the item.

UnitItem:
This subclass of Item has the initStock and current Stock for computeProfit() method where we use the attributes of Stocks to get how many Items have been sold and multiply that by the profit of an individual Item (price - cost) in order to get the profits of all the Items sold. In the sell(int n) method we first check if the n is lesser than the Stock and greater than 0 to compute the price and update the Stock, if n is greater than the Stock the user buys everything.

WeightedItem:
This subclass of Item has the initStock and current Stock for computeProfit() method where we use the attributes of Stocks to get how many Items have been sold and multiply that by the profit of an individual Item (price - cost) in order to get the profits of all the Items sold. In the sell(double n) method we first check if the n is lesser than the Stock and greater than 0 to compute the price and update the Stock, if n is greater than the Stock the user buys everything..

AuctionItem:
This subclass of Item has the bidder to know who is the current Buyer of the Item, deadline to know when the auction has finished. The makeBid(Buyer bidder, double bid, String time) serves to add a bid if the bid is greater than the current one, updating at the same time the bidder, but if the time is greater than the deadline then the auction ends and the bidder buys the Item (we did not know how else should we finish the auction).
In this Item the profit of computeProfit() is the bid minus the fee and tax with the cost, introducing two different terms than the other two subclasses of Item.

Package:
This class has only two attributes, the width and height of the item. being more developed on the subclasses.

Envelope:
This subclass of Package has also the name of the Envelope to differentiate with the name it's size, for example A4 and A3 paper sizes.
The method isSuitable(double[] size) only returns true if the size of the Envelope is bigger than the input size[] but not by more than 5cm, it is assumed that the standard unit size is cm. This is due to thinking that a package too big for what it transports would be an issue.

Box:
This subclass of Package has also a third dimension, depth, that makes it differentiate with other Box instances.
The method isSuitable(double[] size) only returns true if the size of the Envelope is bigger than the input size[] but not by more than 20cm, it is assumed that the standard unit size is cm. This is due to thinking that a package too big for what it transports would be an issue.

Users:
This class is made in order to save the information of those who use the Store. It has three attributes, the name, Id and password, that makes every user different from one another.
The login(String p) method has no user or ID attribute as the usage of it makes it that it is not required since you are trying to log in an instance already, therefore the name and ID are assumed to be the ones of the instance in question.

Buyer:
This subclass of User has an account to be able to buy items and a list with all the items bought.
The buy(Item i, int number) method serves to be able to buy an Item by if it is an UnitItem or WeightedItem updating the Stock and if it is an AuctionItem it will print to screen that as it is an Auction Item you should bid instead.

Seller:
This subclass of User has an account to be able to sell items and a list with all the items to be sold.
The sell(Item i) method serves to be able to sell an item only if it is possible to get the money and the seller has the item in question. After that it checks if there is still stock, else it removes the item from the list of items available.

Admin:
This subclass of User does not introduce new attributes, but has methods that the other subclasses of User do not.
The expel(User u) method checks if the User that is being tried to be expelled is not an Admin, else it expels it. As we do not have a real way of expelling someone this is instead printed on screen.
The manageAuction(AuctionItem i, String d) checks if the Auction is still active or not, if it is not active it prints on screen that the item was managed, else it prints that the auction of the item had ended on the deadline date.

The printStock(List<AuctionItem> i) method prints all the items currently on auction and the most important information, such as the name of the item, the current price and the deadline. Also shows that the administrator is printing the Stock.

The most difficult part of this project was to get the Items and Users to work together. As we had to get all the interactions and update the attributes required, as well as to check if the item was still available or not. At the same time it was quite interesting to do as for that kind of difficulty, we knew how to code and think of how to work around certain problems, but we did not know everything and needed to do research. Though we still do not know how to end the auction without an event to buy or manage the item.