COSC2081- Programming 1: Assignment 2 → 2014C

Auction System

Due date: 9 AM Monday 29/12/2014 (week 12)

Worth: 28% of the overall assessment.

Submission: Zip your Net Beans – Assignment 2 project including the text files and name the zipped archive with your id and name. For example, if your name is Nguyen Van Anh and your id is s1234567, then the zipped archives should be called Assignment2_s1234567_NguyenVanAnh.zip. Upload the zip file of the assignment 2 in the Google Drive and share that in Edit mode on or before 9.00 AM Monday of week 12. I.e. **29/12/2014**. Not submitting in this format will mean your submission will not be graded.

Late submission: Submissions 1 to 5 days late receive 10% penalty per day. Submissions more than 5 days late receive 100% penalty.

Assessment requirements: This is an individual assignment. The minimum penalty for plagiarism is failure for this assignment. Please read http://blackboard.rmit.edu.vn/ → assignments section for further information on School plagiarism policy.

Assignment learning objectives: The aim of this assignment is to help students:

- To be able to code by using object oriented programming concepts

- To be able to demonstrate the concepts of encapsulation, inheritance, abstract classes, exception handling and file handing.

Learning Objectives	RMIT Graduate Attribute	AQF LOD
Enabling knowledge: Using the feature of OOPS- Java The use of standard Java classes	Global in outlook and competence	K1, S1, A1
Problem solving & critical analysis: Ability to design and implement computer programs, code reuse strategies, self-learning on GUI	Innovative	K1, A1
Responsibility: Ability to apply Standards and practices in programming and work individually	Work ready	A1, A3

INTRODUCTION

The scenario we are modeling is simulation of a computerized auction tracking system operated by an auction house that helps customers sell their items in live auctions. Customers can both buy and sell items in the system and sellers can choose to have either an *open* auction, where the highest bid that that equals or exceeds the *starting price* will win the auction when it comes to an end, or a *reserve* auction, where there is a hidden *reserve price* that must be reached before the item if effectively *on the market* (ie. once a bid had been placed which exceeds the hidden *reserve price* the highest bid from that point on will win the auction when it comes to an end).

Auctions will also be *opened* and *closed* manually in the system by administrators of the system when the actual auction is commenced and ended respectively.

In this assignment we are focusing on the tracking of item auction information in the system – the auction company also maintains details about customers who participate in their auctions (both buyers and sellers); these requirements will be discussed in Part III of this assignment. The auction company offers two types of auction to its customers as follows:

- **Open-auctions** in which the item is effectively *on the market* when the auction begins, which means the highest valid bid (which exceeds the *starting price*) will win the item when the auction ends.

- **Reserve-auctions** in which there is a *hidden* reserve price which needs to be met before the item is *on the market* – bids will still be accepted below the *reserve price* (as long as they equal or exceed the *starting price*), but the item will only be sold if the highest bid has equaled or exceeded the *reserve price* that has been set when the auction comes to an end.

The details required for the handling of reserve auction information will be discussed later on in section part II.

You are required to design and implement an object-oriented program that models some basic functionality required in such an auction system.

Part I – Writing an AuctionItem class

 An auction item requires the following details to be maintained in the system: item ID, item description, seller ID, starting price and the auction status (pending, open or closed). Once the auction has been opened the system will also need to maintain the details of the highest bidder and the amount they have bid on the item. You should start by designing and implementing a basic AuctionItem class, as follows:

1. **Define instance variables** for the item ID (an int), item description (a String), seller ID (a String), starting price (a double), auction status (a String which can be "Pending",

"Open" or "Closed"), the current highest bid (a double), and the ID of the current highest bidder (a String).

2. **Define a constructor** for the AuctionItem class which accepts the item ID, item description, seller ID and starting price as parameters and stores the information in the corresponding instance variables.
This constructor should also set the auction status to "Pending" and the highest bid to zero (0). NOTE: You should not have parameters for auction status, highest bid (which are both initialised as discussed in the point above) or the bidder ID (which will be updated when a valid bid is placed on the item).

3. **Define appropriate accessors** for the each of the instance variables in this class.

4. **Define a method public boolean hasBids()**, which returns true if a valid bid has been placed on the item and false if no bid has been made (hint: a valid bid has been made if the current highest bid is equal to or greater than the starting price).

5. **Define a method public double open()**, which functions as described below:

If the current auction status is "Pending" then it changes the status to "Open", calculates the listing fee for the item based on the starting price and returns the result. The price structure for listing fees is as follows:

Starting Price p	Listing Fee
p ≤ $5.00	$0.20
$5 < p ≤ $20.00	$0.50
$20 < p ≤ $ 100.00	$1.00
$100 < p ≤ $250.00	$2.50
p > $250.00	$5.00

If the auction status is not "Pending" then this method should return a result of -1.

6. **Define a method public double close()**, which functions as described below:

If the current auction status is "Open" then it changes the status to "Closed", calculates the sale fee for the item based on the sale price (highest bid) and returns the result. The price structure for sale fees is as follows:

Sale Price p	Sale Fee
Not sold or p < $1.00	$0.00 (no fee)
$1.00 ≤ p ≤ $100.00	5% of sale price
$100 < p ≤ $ 1000.00	$5 + 3% of (sale price – 100)
p > $1000.00	$32 + 1% of (sale price – 1000)

Note: All sale fee amounts should be rounded up to the nearest cent

If the auction status is not "Open" then this method should return a result of -1.

For points 5 and 6 you might find it useful to define separate methods for calculating the listing and sale fees (which you can then call when needed in the **open** and **close** methods), as it will make the changes that will be required for calculating listing and sale fees easier to implement in assignment 4.

7. **Define a method public int placeBid(double bidAmount, String bidder)** which attempts to place a bid on the item being auctioned.
If the auction status is "Open", the bid amount is greater than or equal to the starting price and is also greater than the current highest bid then the bid should be accepted, resulting in the highest bid being set to the new bid amount and the highest bidder being set to the bidder ID that was passed in as a parameter. After the highest bid and bidder information has been updated the method should return the value zero (0) to indicate that the requested bid was successful.

   If the auction status is not "Open" then the method should return a value of -1 to indicate that the auction is not yet open.

   If the auction status is "Open", but the bid amount does not meet the requirement of being greater than or equal to the starting price and also being greater than the current highest bid then the method should return a value of -2 to indicate that the bid was not successful.

8. **Define a method public void print**(), which prints a summary of the details (instance variables) for the current ActionItem to the screen.

   NOTE: The details of the AuctionItem object should be printed in a neat format as shown in the screenshots provided below.

Part I.A – Writing a Test class for AuctionItem class (for demo purpose only)

In this section you need to write an application that creates, stores, and uses a collection of AuctionItem objects. You will need to address the following points in your application class main method:

1. Define an array of AuctionItem references that can store up to four objects (elements).

   Create the AuctionItem objects specified below and store them in the array of AuctionItem references mentioned above. You should pass the values shown below as arguments to the constructor when creating each of the objects.

Item ID	Description	Seller ID	Starting Price
1	Tonka Truck	M001	1.00
2	Xbox	M002	20.00
3	Teddy Bear	M003	5.00
4	Antique Doll	M004	100.00

Note that the sequence and/or format of the item and seller IDs shown above serve as an example only – there is no requirement to ensure that the item IDs or registration numbers are sequential or formatted in any particular way (however, you should use the data provided in the table when creating your objects).

2. Implement a code segment (feature) to display the item ID, description and starting price for of each of the AuctionItem objects, by using a loop to step through each object in the array and invoking the corresponding accessors to retrieve the item ID, description and starting price for each object.

A sample screenshot for this feature is shown below:

```
Available Item List:
Item ID: 1 Desc: Tonka Truck Start Price: $1.00
Item ID: 2 Desc: Xbox Start Price: $20.00
Item ID: 3 Desc: Teddy Bear Start Price: $5.00
Item ID: 4 Desc: Antique Doll Start Price: $100.00
```

3. Open the auction: each of the four items that are being auctioned off by calling the open() method for each AuctionItem object inside a suitable loop.

You should display the listing fee that is returned when the auction is opened to the screen as shown below:

```
Auction 1 has started - listing fee: $0.20
Auction 2 has started - listing fee: $0.50
Auction 3 has started - listing fee: $0.20
Auction 4 has started - listing fee: $1.00
```

4. Implement a bidding feature which allows the user to bid on one of the items that are up for auction. You must allow user to bid as many times as they want to, i.e. this feature must be placed in a loop.

This feature should prompt the user for the AuctionItem ID that corresponds to the item they wish to bid on, after which it will search through the array of AuctionItem objects for one with a matching ID.

If such an object is found then the feature should proceed to prompt the user to enter the bid amount and bidder ID, after which it should attempt to add place a bid on the specified item object by calling the placeBid() method.

If the bid is successful, then status of that item should be changed as "Closed" and the updated details for the AuctionItem object should then be displayed to the screen (as shown below).

If the bid is not successful then a suitable error message should be displayed to the screen.

Note that you also have to report to users if the item is not found.

Unsuccessful bid:

```
Enter the id of the auction to bid on: 4
You are bidding in the auction for an "Antique Doll".
Enter bid amount: 90
Enter bidder id: m001
Invalid bid for auction 4
```

Successful bid:

```
Enter the id of the auction to bid on: 4
You are bidding in the auction for an "Antique Doll".
Enter bid amount: 100
Enter bidder id: m001
Your bid was successful!

Item ID : 4
Description : Antique Doll
Seller ID : m003
Starting price : 100.00        Output format for print() method in
Auction Status : Open          AuctionItem class
Highest Bid : 100.00
Highest Bidder : m001
```

5. Close the auction: each of the four items that are being auctioned off by calling the close()
    method for each AuctionItem object inside a suitable loop.

    You should display the sale fee that is returned when the auction is closed to the screen as
    shown below:

    ```
    Auction 1 has ended - sale fee: $0.00

    Auction 2 has ended - sale fee: $0.00

    Auction 3 has ended - sale fee: $0.00

    Auction 4 has ended - sale fee: $5.00
    ```

    *Note: These figures are what you would see after the successful bid scenario shown on the previous page
    has occurred.*

PART II – Writing an ReserveItem Class

In Part I, you already implemented AuctionItem class that represents *open auction items*. In this section, you will implement the handling of *reserve auction items*.

> In **reserve-auctions** there is a *hidden reserve price* which needs to be met before the item is *on the market* – bids will still be accepted below the *reserve price* (as long as they equal or exceed the *starting price*), but the item will only be sold if the highest bid has equaled or exceeded the *reserve price* that has been set when the auction comes to an end.

You are required to design and implement an object-oriented program that models some basic functionality required in such an auction system.

ReserveItem class

A *reserve auction item* requires the *reserve price* set for the auction to be maintained (in addition to the details that are already maintained for an open auction) - this *reserve price* will determine if the item is *on the market* (i.e. whether the item will be sold to the highest bidder or not). The *reserve price* is also used to determine *listing fees* for reserve auctions and additionally if factored into *sale fee* calculations when the reserve auction item is sold. You should address this new requirement by designing and implementing a ReserveItem class (extending the AuctionItem class of Assignment 3), which includes additional features as described below:

1. **Define a new instance variable** for the *reserve price* (a double).

   You should not redefine any of the instance variables defined initially in the AuctionItem superclass in this ReserveItem subclass.

2. **Define a constructor** for the ReserveItem class which accepts the item ID, item description, seller ID, starting price and reserve price as parameters and stores the information in the corresponding instance variables.

   You should use the super() facility to pass on the relevant parameters (item ID, description, seller ID and starting price) to the superclass constructor. As was the case with the AuctionItem class constructor you do not need to pass parameters for the auction status, highest bid or bidder ID as these will be initialized to default values in the AuctionItem superclass constructor.

3. **Override the accessor** for auction status from the AuctionItem superclass so that it functions in the following manner for a ReserveItem:

   If the auction status is "Closed", the item has had a valid bid placed upon it and the current highest bid is less than the reserve price then this overridden accessor should return "Passed In", otherwise this method should return the result obtained by calling the

accessor for auction status from the AuctionItem superclass (there is no need to store the "Passed In" state anywhere).

4. **Override the functionality for determining listing fees** based on the *reserve price* that has been set for the auction according to the price structure shown in the table below:

Reserve Price r	Listing Fee
r ≤ $1.00	No listing fee
$1.00 < r ≤ $100.00	4% of the reserve price
$100 < r ≤ $1000.00	$4 + 2% of (reserve price - $100)
r > $1000.00	$22 + 3% of (reserve price – $1000)

If you calculated the *listing fee* in a separate method in the AuctionItem superclass then you could override that method in this subclass without having to make any change to or override the open() method in this subclass. Otherwise you will need to override the entire open() method in this subclass.

5. **Override the functionality for determining sale fees** so that it calculates and returns a result of 2% of the *reserve price* if the auction item has been *passed in* (you can use the overridden auction status accessor to help with checking if the item was *passed in* or not).

If the auction item status is not "Passed In" then the *sale fee* should be calculated according to the price structure described for the AuctionItem superclass - you can just call the superclass version of the either the method which calculates the sale fee (or the superclass version of the close() method) and pass on the result it returns in this situation.

If you calculated the sale fee in a separate method in the AuctionItem superclass then you could override that method in this subclass without having to make any change to or override the close() method in this subclass. Otherwise you will need to override the entire close() method in this subclass.

> NOTE: You will need to use the `super` "reference" to invoke the method for calculating the sale fee (or the close() method if you are overriding it in the subclass) from the `AuctionItem` superclass in order to perform the basic sale fee calculation if it is required, after which you can return the result that is returned (which will be the sale fee).

6. **Define a method public boolean lowerReserve(double newReservePrice)**, which can be used to lower the *reserve price* for a ReserveItem so that the item can be sold if the seller decides to try to encourage more bidding by lowering the *reserve price* that has been set.

The *reserve price* for an item can only be lowered if the auction is currently in progress (status is "Open") and the *new reserve price* must be less than the *current reserve price* that has been set. If both of these conditions have been met then the *current reserve price* should be set to the *new reserve price* and the method should return a true result, otherwise it should return a false result.

7. **Override the print()** method from the AuctionItem superclass so that it also prints the reserve price for a ReserveItem.

NOTE*: The overriding version of the `print()` method in the `ReserveItem` class will need to use the "`super`" reference to invoke the corresponding method from the `AuctionItem` superclass, in order to display the `AuctionItem` details first. The details of the `ReserveItem` object should be printed in a neat format (an example is shown in the screenshots provided later on in this document).*

PART III – Writing a Customer class with its subclasses

This section is an exercise in object-oriented programming and as such you should adhere to the following guidelines when designing and writing your classes:

a) Set the visibility of all instance variables to private.
b) Encapsulate both the data and the methods of classes within the same class.
c) Use superclass methods to manipulate superclass data from within a subclass, where necessary.

The assignment continues with the theme of Part I & II, that of the auction tracking system, for an auction house which runs both *open auctions* (with no set reserve price) and *reserve auctions* (where the seller sets a minimum price they are willing to sell their item for). The auction house caters to both *casual customers* as well as *registered customers* who can buy and sell items in auctions run by the auction house.

In Part 1 and 2 we focused on developing a framework for recording information about *items* put up for auction, as well as recording *bids* made in those auctions. In assignment 3- part III onwards we will be focusing on developing the framework for recording and maintaining customer details in the system, as well as completing the implementation of the required auction-related functionality in the system. Customers are categorized in two ways: (a) **casual customers**, who are can only *place bids on auctions*, and (b) **registered customers** who are permitted to *place items up for auction*, as well as *place bids on any auctions* they wish to participate in. *Casual customers* are given a Personal Identification Number (PIN) which they use to verify their identity when paying for items they have won at auction – they also have a credit card number recorded which is used to organize payment for any auctions they win. *Registered customers* go through a more formal registration process and thus are granted access to a line of credit within the auction tracking system, from which auction listing and sale fees are taken for any items they wish to sell – payment for any auctions they win are also made from this credit account. *Registered customers* can top up their credit account balance at any time and are only permitted to place new items up for auction if they have settled any existing amount they owe from previous auctions the auction house has run for them and/or payments stemming from auctions they have won. Only placing items up for auction is affected by this restriction – *registered members* are still free to bid on item auctions and listing/sale fees can still be deducted from their account even if the *registered customer* currently owes money to the auction house.

Modeling customer details

1. Customer class

There are common details which need to be recorded for both casual and registered customers, specifically a (unique) customer ID and the customer's name.

1. Define instance variables for the customer ID (a String) and the customer name (a String). Also define a constructor for the Customer class, which accepts the customer ID and name as parameters.

2. Define accessors for the customer ID and name.

3. Define an abstract method **public abstract void completeSale (double price)**, which the customer subclasses will have to override later on.

4. Define a method **public void print**(), which prints a summary of the details (instance variables) for the customer to the screen.

2. CasualCustomer class

A CasualCustomer can place bids on items that are up for auction, but as a casual auction participant they are not permitted to place items up for auction. A credit card number is verified and recorded for casual customers and they are given a Personal Identification Number (PIN) which they can use to verify their identity when making payment for an item they have won at auction. You should address these new requirements by designing and implementing a CasualCustomer class which extends the basic Customer class to cater for these additional requirements.

1. Define new instance variables for the credit card number and PIN (Personal Identification Number) recorded in the system for the new CasualCustomer.
   NOTE: You should not redefine any of the instance variables defined initially in the (Customer) superclass in this (CasualCustomer) subclass.

   Also define a constructor which accepts the customer id, name, credit card number and PIN for the new CasualCustomer as arguments.
   NOTE: You should use the super() facility to pass on the relevant arguments (ID and name) to the superclass constructor.

2. Define an accessor for the credit card number (an accessor for the PIN is not required).

3. Override the method **completeSale(double price)** so that it prompts the customer for their PIN and checks it against the PIN stored for them in the system.
   If the PIN entered by the user is a match then a suitable message stating that the sale price has been charged to their credit card (number) should be printed to the screen. If the PIN entered by the user does not match the one stored in the system then a message should be printed to the screen, stating that the item has been withheld pending confirmation of the customer's identity.

4. Override the **print()** method for this subclass so that it prints a summary of all of the details (instance variables) for the current CasualCustomer to the screen.
NOTE: The overriding print() method in the CasualCustomer class will need to invoke the corresponding method from the (Customer) superclass to print the basic Customer details first.

3. RegisteredCustomer class

A RegisteredCustomer is granted access to a credit account in the auction tracking system from which auction listing/sale fees are deducted – this credit account is also used to pay for any items the customer wins at auction. The balance of this credit account can go into the negative to reflect that the customer owes the auction house money for fees charged on auctions they have had run on their behalf or for purchases made. You should address these new requirements by designing and implementing a RegisteredCustomer class which extends the basic Customer class.

1. Define a new instance variable for the account balance (a double).
NOTE: You should not redefine any of the instance variables defined initially in the (Customer) superclass in this (RegisteredCustomer) subclass.

   Also define a constructor which accepts the customer ID, name, and starting account balance for the new RegisteredCustomer as arguments. If the value passed in for the starting balance is less than zero then it should be set to zero by default.
NOTE: You should use the super() facility to pass on the relevant arguments (customer ID and name) to the superclass constructor.

2. Define an accessor for the account balance.

3. Implement a method public void addFunds(double amount)which adds the specified amount to the account balance for the current RegisteredCustomer.

4. Implement a method public void deductFees(double fees) which deducts the specified fee amount from the account balance for the current RegisteredCustomer.
   NOTE: It is acceptable for the account balance to go into the negative if needs be.

5. Override the method public void completeSale(double price) so that it deducts the specified sale price from the account balance of the RegisteredCustomer (it is acceptable for the account balance to go into the negative if needs be).

6. Override the method public void print() so that it prints a summary of all of the details (instance variables) for the current RegisteredCustomer to the screen.
NOTE: The overriding version of the print() method in the RegisteredCustomer class will need to invoke the corresponding method from the (Customer) superclass to print the basic Customer details first.

Part IV– Exception Handling

Update the open(), close() and placeBid() methods in the AuctionItem class (and the ReserveItem class if you have overridden the open() and close() methods in the subclass) so that they throw exceptions in all cases where error signals are currently being returned (i.e. the auction is not in the correct state to be opened/closed/bidded on, bid amount is invalid, etc).

- open() returns the listing fee. If the auction status is not "Pending" then this method should throw an exception instead of returning -1.

- close() returns the sale fee. If the auction status is not "Open" then this method should throw an exception instead of returning -1.

- placeBid() must not return anything. If there is an error about auction status or bid amount then this method should throw an exception instead of returning -1 or -2.

The exceptions that are thrown should contain an appropriate error message and should be caught in an appropriate fashion in the application class (at which point the error message stored in exception object should be printed to the screen).

You can either define your own exception type(s) or use the basic Exception type when throwing exceptions where necessary in the AuctionItem/ReserveItem classes (either way is fine). You can also throw exceptions in the lowerReserve() method of the ReserveItem class if you wish to, but this is not a requirement and you will not be penalised for not doing so.

Part V - File Handling

1. Writing of item and customer data to text file(s)

Your program should implement a feature for writing all item and customer details out to file. The format of the files data is written out to is up to individual students, but the files themselves must be text files (i.e. any method which involves writing data out in binary form is not permitted). This file writing feature should be run when the user chooses to exit the program.

2. Reading of item and customer data from text file(s)

Your program should implement a feature for reading all item and customer information into the system from the data file/s created by the file writing feature discussed above. The item and customer information that is read in from the file should be used to create an appropriate set of objects for each set of data which should be stored in the corresponding arrays when the program first starts running. If a file cannot be opened for reading then the program should print a suitable message stating that the file was not read and the program will continue without loading information for items and/or customers to the screen. In order to receive marks for reading item/customer information from file your file reading feature must be compatible with

the format of the files created by file writing feature discussed above. This file reading feature should be run when the program is first started.

Part VI - Writing a menu driven application that using AuctionItem, ReserveItem, and Customer classes

For this part of the assignment you need to write a menu-driven application that allows the user to create a mix of AuctionItem and ReserveItem objects to represent new auctions for items, open/close those auctions, place bids on the items being auctioned and display the details for all items in the auction system. You will need to address the following points in your application class:

1. Storage for AuctionItem and ReserveItem objects

Define an array of AuctionItem references that can store up to ten objects (elements). This must be a single array – you cannot use separate arrays for AuctionItem and ReserveItem objects.

2. Storage for CasualCustomer and RegisteredCustomer objects

Define an array of Customer references that can store up to 50 objects (elements) - it may be an advantage to declare and instantiate this array as a static member of your application class (i.e. before of the main method). This must be a single array – you cannot use separate arrays for CasualCustomer and RegisteredCustomer objects.
Note: You should also change the size of the array of AuctionItem references created as part of assignment 4 so that it can also store up to 50 objects (elements).

3. Program Menu

Implement a menu feature which presents options for adding a new auction, displaying details of all auctions in the system, opening an auction for bidding, placing a bid, closing an auction and exiting the program as shown below:

```
*** iBuy Auction Recording System *********

A - Add Auction
B - Display Auction Details
C - Open Auction
D - Place Bid
E - Close Auction
F - Add Customer
G - Display Customer Details
H - Add Funds to Account
X - Exit the Program
Enter you selection:
```

After processing the user's selection the program should return to the menu until they select the option to exit the program. A suitable error message should be displayed and the program should display the menu/prompt for a selection again if the user enters a selection outside of the valid range of menu options (i.e. the program should not terminate if the user enters an invalid selection).

4. **Add Auction feature**

This feature should prompt the user to enter all basic auction details (item id, description, starting price, seller id). If the item ID is already present in the array then the feature should display a suitable error message and the program should return immediately to the menu. If the duplicate item id check is passed then the user should be prompted to select the type of auction they wish to add to the system - if they choose a basic auction then a new AuctionItem object should be created (passing in the details supplied by the user) and added the next vacant position in the array of AuctionItem references mentioned above. If the user chooses to start a reserve auction then they should be prompted to enter a reserve amount and a new ReserveItem object should be created (passing in the details supplied by the user) and added to the next vacant spot in the array of AuctionItem references mentioned above. Screenshots demonstrating how this feature could function are shown below:

```
******** iBuy Auction Recording System *********

A - Add Auction
B - Display Auction Details
……
……
……
X - Exit the Program

Enter you selection: A
Adding a new item auction...
Enter item ID: 1
Enter item description: Teddy Bear
Enter seller ID: Halil3
Enter starting price: 5.00
Create a new AuctionItem or a ReserveItem? (enter A or R): A
Auction item "Teddy Bear" successfully added to system!

******** iBuy Auction Recording System *********

A - Add Auction
B - Display Auction Details
……
……
……
X - Exit the Program

Enter your selection: A
Adding a new item auction...
Enter item ID: 2
Enter item description: Antique Doll
Enter seller ID: Jessica9
Enter starting price: 5.00
Create an AuctionItem or a ReserveItem? (enter A or R): R
Enter reserve price: 100.00
```
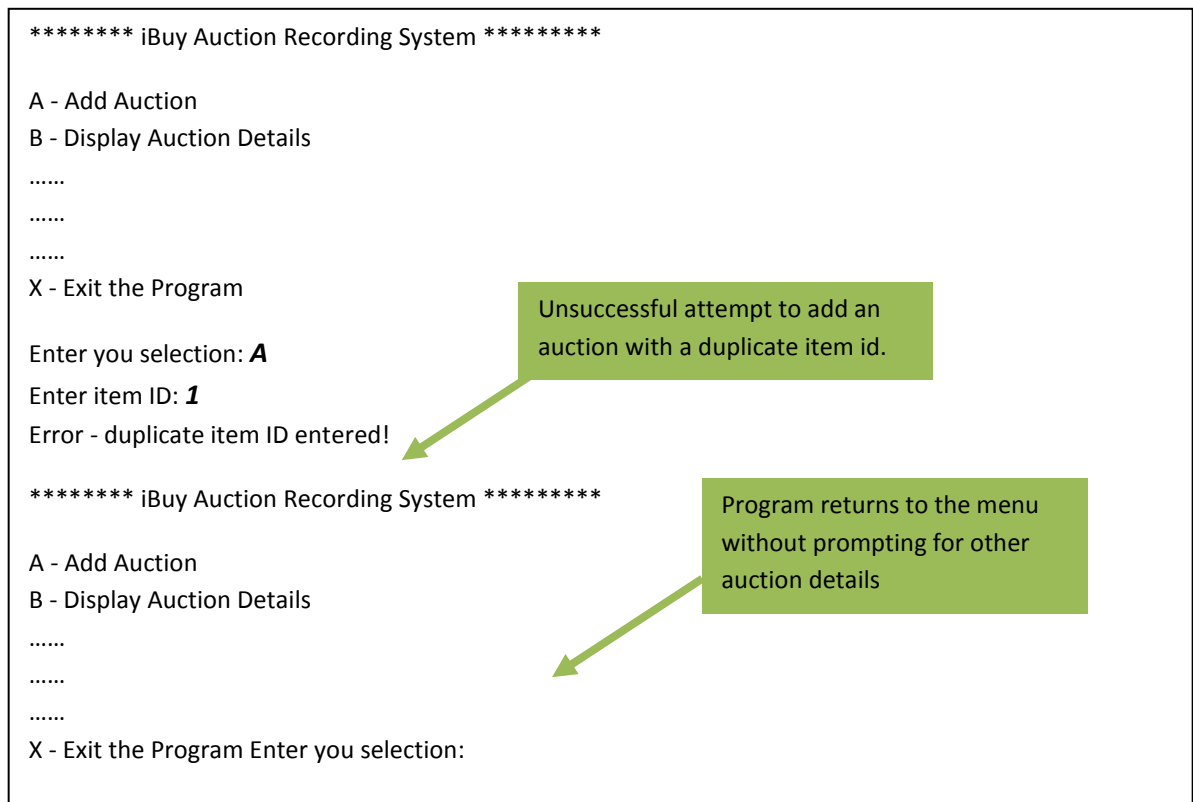
Successful attempt to add an AuctionItem

Successful attempt to add a ReserveItem

******** iBuy Auction Recording System ********

A - Add Auction
B - Display Auction Details
……
……
……
X - Exit the Program

Enter you selection: *A*

Enter item ID: *1*

Error - duplicate item ID entered!

******** iBuy Auction Recording System ********

A - Add Auction
B - Display Auction Details
……
……
……
X - Exit the Program Enter you selection:

> Unsuccessful attempt to add an auction with a duplicate item id.

> Program returns to the menu without prompting for other auction details

Note: The add auction feature should be updated so that it locates the customer who is selling the item after the seller ID has been entered by the user - if a customer object is not found with the specified ID then the feature should print a suitable error message to the screen and return to the menu.

If a customer object with the specified ID is found then that object should be examined to see if it is a RegisteredCustomer object – if it is not then the feature should print a suitable error message to the screen and return to the menu.

If the customer object with the specified ID is a RegisteredCustomer then the account balance for that RegisteredCustomer object should be checked – if the account balance is negative (less than zero) then the feature should print a suitable error message to the screen and return to the menu.

If all of these checks are passed the feature should proceed with the process of adding of a new item as described above.

5. **Display Auction Details feature**

This feature should display all details for each auction in the system by using a loop to step through the array and invoke the print() method for each object that is currently being stored. A sample of the output that might be produced by this feature is shown below.

```
******** iBuy Auction Recording System *********

A - Add Auction
B - Display Auction Details
……
……
……
X - Exit the Program

Enter you selection: B
Printing details of all auctions in the system:

Item ID: 1
Description: Teddy Bear
Seller ID: Halil3
Starting price: 5.00
Auction Status: Pending

Item ID: 2
Description: "Antique Doll"
Seller ID: Jessica9
Starting price: 5.00
Auction Status: Pending
Reserve Price: $100.00
```

This is the output that would be displayed after auctions shown in the previous section were added.

6. **Open Auction feature**

This feature should prompt the user to enter the ID of the auction they wish to open and proceed to search for an item (auction) in the array of AuctionItem references that has the item id specified by the user. If a matching item (auction) object is not found then a suitable error message should be printed to the screen. If matching item (auction) object is found then the feature should call the open() method for that object in an appropriate fashion - if the call to open() is successful then the listing fee should be displayed to the screen, otherwise a suitable error message should be printed to the screen.. A sample of the output that might be produced by this feature is shown below.

```
******** iBuy Auction Recording System ********

A - Add Auction
B - Display Auction Details
C - Open Auction
……
……
X - Exit the Program

Enter you selection: C
Enter the id of the auction to open: 3
The auction item with id 3 is not found!

******** iBuy Auction Recording System ********

A - Add Auction
B - Display Auction Details
C - Open Auction
……
……
X - Exit the Program

Enter you selection: C
Enter the id of the auction to open: 2
Auction 2 has started - listing fee: $4.00

******** iBuy Auction Recording System ********

A - Add Auction
B - Display Auction Details
C - Open Auction
……
……
X - Exit the Program

Enter you selection: C
Enter the id of the auction to open: 2
Auction 2 cannot be opened because it is not pending!
```

Non-existent item id entered

Auction opened successfully

Attempt to open auction failed as auction for that ID is not pending (it was opened previously in this case)

Note: The open auction feature should be updated so that it locates the RegisteredCustomer who is selling the item. Once the corresponding RegisteredCustomer object has been located, the listing fee (which is returned when the open() method is called) should be deducted from their account balance by calling the deductFees() method in an appropriate fashion. Exceptions thrown by open() must be handled here.

Note: As the `deductFees()` method is a subclass-only method you may find the `instanceof` operator and typecasting useful for this update to the open auction feature. There is also no need to handle situations where the customer ID is not found or the matching object is not a `RegisteredCustomer` object as these situations cannot occur due to the validation that takes place when an item is put up for auction.

7. **Place Bid feature**

Implement a feature which allows the user to place a bid on an active (open) auction. This feature should prompt the user to enter the ID of the auction they wish to bid and proceed to search for an item (auction) in the array of AuctionItem references that has the item id specified by the user. If a matching item (auction) object is not found then a suitable error message should be printed to the screen. If matching item (auction) object is found then the feature should prompt the user to enter a bid amount and a bidder id, after which it should call the placeBid() method for that object in an appropriate fashion. If the call to placeBid() is successful then the details for the item (auction) object should be displayed to the screen by calling the print() method in an appropriate fashion. If the call to place bid is not successful (result returned is < 0), then a suitable error message should be printed to the screen - you may identify specific issues if you wish to do so (i.e. auction is not open, bid amount is invalid, etc). A sample of the output that might be produced by this feature is shown below.

```
******** iBuy Auction Recording System *********

A - Add Auction
……
……
D - Place Bid
----                                    Non-existent item id entered
X - Exit the Program

Enter you selection: D
Enter the id of the auction to bid on: 3
The auction item with id 3 is not found!

******** iBuy Auction Recording System *********

A - Add Auction
……
……
D - Place Bid
----                                    Invalid bid placed for auction
X - Exit the Program

Enter you selection: D
Enter the id of the auction to bid on: 2
You are bidding in the auction for item "Antique Doll".
Enter bid amount: 1.00
Enter bidder id: craig2
Invalid bid for auction id 2
```
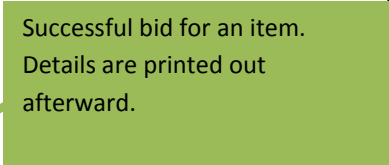
```
******** iBuy Auction Recording System *********

A - Add Auction
……
……
D - Place Bid
----
X - Exit the Program

Enter you selection: D
Enter the id of the auction to bid on: 2
You are bidding in the auction for item "Antique Doll".
Enter bid amount: 95
Enter bidder id: daryl4
Your bid was successful!

Item ID : 2
Description : Antique Doll
Seller ID : Jessica9
Starting price : 5.00
Auction status : Open
Highest bid : 95.00
Highest bidder : daryl4
Reserve price : $100.00
```

Successful bid for an item. Details are printed out afterward.

Note: The place bid feature should be updated so that it verifies that the bidder ID entered by the user is an actual ID for one of the customers that have been added to the system – if a customer object with the specified ID is not found then the feature should print a suitable error message to the screen and return to the menu. If a matching customer object is found then the feature should proceed with process of placing of a bid as described in assignment 4. Exceptions thrown by placeBid() must be handled here.

8. **Close Auction feature**

   This feature should prompt the user to enter the ID of the auction they wish to close and proceed to search for an item (auction) in the array of AuctionItem references that has the item id specified by the user. If a matching item (auction) object is not found then a suitable error message should be printed to the screen. If matching item (auction) object is found then the feature should attempt to open the auction for that item by calling the close() method for that object in an appropriate fashion - if the call to close() is successful then the listing fee should be displayed to the screen, otherwise a suitable error message should be printed to the screen.

   Note: this process is very similar to that for the Open Auction feature

   A sample of the output that might be produced by this feature is shown below.

   ```
   ******** iBuy Auction Recording System *********

   A - Add Auction
   B - Display Auction Details
   -------
   -------
   E - Close Auction
   X - Exit the Program

   Enter you selection: E

   Enter the id of the auction to close: 2

   Auction 2 has ended - sale fee: $2.00
   ```

Note: The close auction feature should be updated so that it checks to see if the item object is a ReserveItem before attempting to call the close() method – if it is and the current highest bid is below the reserve price then the user should be prompted to indicate whether they wish to lower the reserve price (so that the item can sell). If the user indicates they wish to lower the reserve price then the feature should prompt for the new reserve price and the feature should then proceed to call the lowerReserve() method for the ReserveItem object in an appropriate fashion (passing along the new reserve price entered by the user). Exceptions thrown by lowerReserve() and close() must be handled here.

```
******** iBuy Auction Recording System *********

A - Add Auction
B - Display Auction Details
-----
-----
-----
X - Exit the Program

Enter you selection: B
Item ID : 1
Description : Teddy Bear
Seller ID : Halil3
Starting price : 5.00
Auction Status : Pending

Item ID : 2
Description : Antique Doll
Seller ID : Jessica9
Starting price : 5.00
Auction Status : Passed In
Highest Bid : 95.00
Highest Bidder : Daryl4
Reserve Price : $100.00

******** iBuy Auction Recording System *********

A - Add Auction
-----
-----
-----
E - Close Auction
X - Exit the Program

Enter you selection: E
Enter the id of the auction to close: 3
The auction item with id 3 is not found!

******** iBuy Auction Recording System *********

A - Add Auction
-----
-----
-----
E - Close Auction
X - Exit the Program

Enter you selection: E
Enter the id of the auction to close: 2
Auction 2 cannot be close because it is not currently open!
```

The state of the (ReserveItem) object that was closed has now changed!

Item not exists

Attempt to open auction failed as auction for that ID is not open (it was closed previously in this case)

9. Add Customer

This feature should prompt the user to enter basic customer details (customer id and name).

If the customer ID is already present in the array then the feature should print a suitable error message to the screen and return to the menu.

If the duplicate customer id check is passed then the user should be prompted to select the type of customer they wish to add to the system. If they choose to add a *casual customer* then the user should be prompted to enter a credit card number and PIN and a new CasualCustomer object should be created (passing in the details supplied by the user), which should then be added the next vacant position in the array of Customer references mentioned above. If the user chooses to add a *registered customer* then they should be prompted to enter a starting account balance and a new RegisteredCustomer object should be created (passing in the details supplied by the user), which should then be added to the next vacant spot in the array of Customer references mentioned above.

10. Feature: Display Customer Details

This feature should print all details for each customer in the system by using a loop to step through the array and invoke the print() method for each customer object that is currently being stored.

11. Feature: Add Funds To Account

This feature should prompt the user to enter the ID of the customer whose account is to have funds added to, after which the feature should proceed to locate a customer with the specified ID in the array of Customer references mentioned above. If an object with the specified customer ID is not found then the feature should print a suitable error message to the screen and return back to the menu. If an object with the specified customer ID is found then that object should be examined to see if it is a RegisteredCustomer object – if it is not then the feature should print a suitable error message to the screen and return to the menu.
Hint: You may find the instanceof operator useful here.

If the object that was found is a RegisteredCustomer object then the user should be prompted to enter the amount to add to the account balance for the RegisteredCustomer and call the addFunds() method for the RegisteredCustomer object (passing in the amount specified by the user).

Hint: You will need to typecast your `Customer` reference in order to call the `addFunds()` method for the `RegisteredCustomer` object that was found.

OTHER ASSESSMENT CRITERIA – CODING STYLE

- Indentation levels consistent throughout program
- Lines of code not exceeding 80 characters in length - lines which will exceed this limit are split into two or more segments where required
- Expressions well-spaced out
- Source spaced out into logically related segments
- Use of appropriate/descriptive identifiers wherever possible to improve code readability
- Use of comments to describe the purpose of each data class, each method within a data class and any non-trivial segments of code within those methods.

FINAL NOTES ON CODE RE-USE

This assignment is an exercise in code-reuse as much as anything else – while it might look like there is quite a bit to do in the application class, many elements are similar across all of the features (e.g. searching for an item is always the same and is similar to what was required for assignment part I). There will be elements you can take from part I (either your own or the sample solution). There will also be times when you can take advantage of methods in the superclass to help you perform tasks in methods you are writing for the subclass, as well as times when you can "re-use" code you have written for a completed feature to help code the feature you need to take care of next in the application class. As we progress in Programming 1 it is expected that you take advantage of code re-use as much as possible, but in truth doing so only helps the programmer (you) in the long run as it saves you time and effort from not having to re-write the same code segments from scratch over and over again.

Suggested schedule

It is advisable to start working on this assignment as soon as possible. Leaving the assignment until the week before the deadline will make it very difficult for you to complete the assignment on time.

The following is a suggested list of internal dates/deadlines so that you can measure you progress:

Week	Activities
6	Download assignment, read documentation start building classes – Part I & Part II
7	Complete all class skeletons (Part I, part II and Part III) and add required properties to all model classes. i.e. AuctionItem, ReservedItem, Customer, CasualCustomer, and RegisteredCustomer
8	Create methods for reserveItem and Customer classes
9	Demo – AuctionItem , Test class, ReserveItem,Customer and its subclasses Create main menu code in Main class with main() method, implement methods to following menu options A - Add Auction B - Display Auction Details C - Open Auction D - Place Bid E - Close Auction F – Add Customer G – Display Customer Details H – Add Funds to Account X - Exit the Program
10	Add required constructors to all classes, implement Exceptions, implement exception handling
11	Check your work and fix any small problems

ASSIGNMENT 2 COMPLETION REPORT / MARKING GUIDE

Student name & ID: _____

Items	Check if completed	Possible points	Actual points
AuctionItem class – Part I (15 marks) *			
Instance and class variables		1	
Constructor		2	
Accessors and Mutators		1	
hasBids()		1	
Open(), preferably with a separate listingFee() method		2	
Close(), preferably with a separate saleFee() method		2	
placeBid()		2	
Print()		1	
Test class for AuctionItem class		3	
ReserveItem class - Part II (10 marks) *			
Instance variable & constructor		1	
Accessor - overriding		2	
Listing fee calculations- overriding		2	
Sale fee calculation - overriding		2	
lowerReserve()		2	
print() – overriding		1	
Customer class – Part III (5 marks) *			
Instance variables		1	
Constructor		1	
Accessors and mutators		1	
completeSale () – abstract method definition		2	
CasualCustomer class (5 marks) *			
Instance variable, constructor and accessor		2	
completeSale(double price) – override (abstract method)		2	
print() – overriding		1	
RegisteredCustomer class (10 marks) *			
Instance variable & constructor & accessor		2	
addFunds(double amount)		2	
deductFees(double fees)		2	
completeSale(double price) – override (abstract method)		3	
print() – overriding		1	
Sections that were * marked should be demonstrated in week 10. Demo schedule will be released in week 9 in the blackboard. 20 marks will be deducted from the total (assignment 2) if student fails to show up for demo session.			

Implementation of classes – Application features: Part VI (31 marks)			
Menu - design		1	
Add Auction		5	
Display Auction Details		2	
Open Auction		5	
Place Bid		5	
Close Auction		5	
Add Customer		3	
Display Customer Details		2	
Add Funds to Accounts		2	
Exit the program		1	
Exception Handling - Part IV(6 marks)			
Open an auction		2	
Close an auction		2	
Place bid		2	
File Handling - Part V (10 marks)			
Writing to files		5	
Reading from files		5	
Coding conventions (8 marks)			
Indentation		2	
comments		2	
Coding techniques		4	
Total		100	