HOMEWORK 6- Fall 2020

HOMEWORK 6 - due Tuesday, November 17th no later than 5:00PM

REMINDERS:

- **Be sure your code follows the coding style for CSE214.**
- **Make sure you read the warnings about academic dishonesty. *Remember, all work you submit for homework or exams MUST be your own work.***
- **Login to your grading account and click "Submit Assignment" to upload and submit your assignment.**
- **You may use (and are encouraged to use) any Java API Data Structures you like to implement this assignment.**
- **You may use Scanner, InputStreamReader, or any other class that you wish for keyboard input.**

When working with data, we must choose the best data structure to manipulate the data set in order to achieve efficient results. A very basic array can store information without a problem, but is inefficient when we are constantly storing and searching for data. The idea of a table is created to increase the efficiency and avoid such problems. Tables store data using a tuple <Key, Value>. We choose a special field of an object, and use it to uniquely identify the entire object. Therefore, when we store and search for an object, we will use the key to save or search for the object. Ideally, this will reduce the algorithm to O(1). For this assignment, you are allowed to use a HashTable, or a HashMap.

For this assignment, we are working with a real auction data online. The data will be in the form of an XML file. There may be extraneous data that we do not need in our analysis. We will selectively limit our object construction with information that we are interested in. To get started, visit the following websites:

Ebay Auction Data

Yahoo Auction Data

Since these URLs are long, we provide shortcuts via the correspnding TinyURLs:

http://tinyurl.com/nbf5g2h - Ebay Auction Data

http://tinyurl.com/p7vub89 - Yahoo Auction Data

Manually extracting the data will be a tedious and error prone job. You will learn how to use a data extraction tool to analyze an XML file. You can download the file here: Big Data Jar. In addition to the sample code provided below, there is documentation available here: Big Data Documentation.

USING A JAR IN ECLIPSE:

Right click the project name in the “Package Explorer” tab (on the left, by default) à Select “Build Path” à Select “Add External Archives…” à navigate to where you saved bigdata.jar and select it.

USING A JAR IN NETBEANS:

Right click the project name in the “Package Explorer” tab (on the left, by default) à Click on “Properties” à Select “Libraries” on the leftà Click on "Add JAR/Folder" on the rightà navigate to where you saved bigdata.jar and select it.

Now you can “`import big.data.DataSource`” in your source code (or any other class from the big.data library that you need).

Note: DO NOT SUBMIT bigdata.jar along with your Java source files, otherwise 10 points will be deducted.

If we have the URL of the XML file, we can connect to it using the library functions. First, we must `connect()` to the URL, then `load()` the data. Finally, we can `fetch()` the information.

Sample Code:

The file <sample_file.xml> contains the following content:

```
<root>
   <item>
      <attr>1</attr>
   </item>
   <item>
      <attr>2</attr>
```

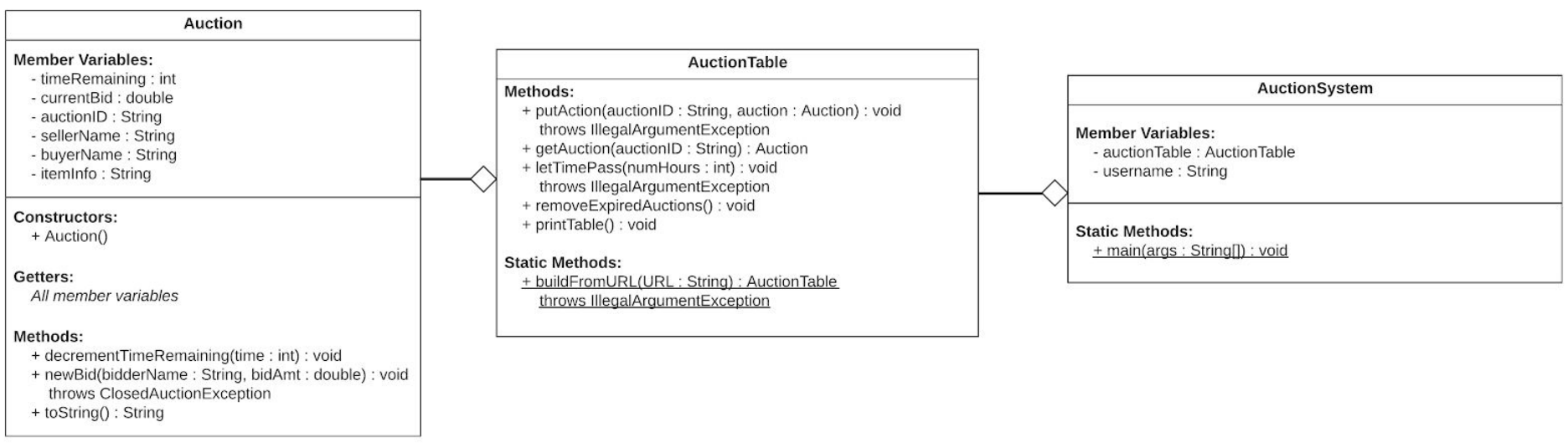
```
  </item>
</root>

import big.data.*;
DataSource ds = DataSource.connect("sample_file.xml").load();
String str = ds.fetchString("item/attr");
// normally, we can extract a single string data in the path item/attr, but we have multiple values in our
case. Instead,
we should do the following:
String[] myList = ds.fetchStringArray("item/attr");
```

You will mostly use `fetchStringArray()` in this assignment. Sometimes, the information on the XML may not be complete. You might find that your fetch will return an empty string (i.e. ""). In that case, you can replace it with the string: `"N/A"`.

Required Classes

The following sections describe classes which are required for this assignment. Each section provides a description and the specifications necessary to complete each class. If you feel that additional methods would be useful, feel free to add them during your implementation as you see fit. However, all the variables and methods in the following specifications must be included in your project.

NOTE: All classes listed should implement the `Serializable` interface.



UML Diagram.

1. Auction

Write a fully-documented class named `Auction` which represents an active auction currently in the database. The `Auction` class should contain member variables for the seller's name, the current bid, the time remaining (in hours), current bidder's name, information about the item, and the unique ID for the auction. In addition, the class should implement a `toString()` method, which should print all of the data members in a neat tabular form.

The `Auction` class can only be altered by a single member method called `newBid()`, which takes in the name of a bidder and their bid value. This method checks to see if the bid value is greater than the current bid, and if it is, replaces the current bid and buyer name. There should be getter methods for each member variable, however, no setters should be included.

The following is a list of information you should be paying attention to:

```
listing/seller_info/seller_name
listing/auction_info/current_bid
listing/auction_info/time_left
listing/auction_info/id_num
listing/auction_info/high_bidder/bidder_name
// the following should be combined to get the information of the item
listing/item_info/memory
listing/item_info/hard_drive
listing/item_info/cpu
```

- **`public Auction()` - constructor (you may include a constructor with parameters)**
- **An `int` member variable:**
  - **`timeRemaining`**
- **A `double` member variable:**
  - **`currentBid`**
- **Four `String` member variables:**
  - **`auctionID`**
  - **`sellerName`**
  - **`buyerName`**
  - **`itemInfo`**

- `public void decrementTimeRemaining(int time)`
  - ***Brief:***
    - **Decreases the time remaining for this auction by the specified amount. If `time` is greater than the current remaining time for the auction, then the time remaining is set to 0 (i.e. no negative times).**
  - ***Postconditions:***
    - **`timeRemaining` has been decremented by the indicated amount and is greater than or equal to 0.**
- `public void newBid(String bidderName, double bidAmt) throws ClosedAuctionException`
  - ***Brief:***
    - **Makes a new bid on this auction. If `bidAmt` is larger than `currentBid`, then the value of `currentBid` is replaced by `bidAmt` and `buyerName` is replaced by `bidderName`.**
  - ***Preconditions:***
    - **The auction is not closed (i.e. `timeRemaining > 0`).**
  - ***Postconditions:***
    - **`currentBid` Reflects the largest bid placed on this object. If the auction is closed, throw a `ClosedAuctionException`.**
  - ***Throws:***
    - **`ClosedAuctionException`: Thrown if the auction is closed and no more bids can be placed (i.e. `timeRemaining == 0`).**
- **`public String toString()` - returns string of data members in tabular form.**
- **`Getters for all data members (no setters).`**

2. AuctionTable

The database of open auctions will be stored in a hash table to provide constant time insertion and deletion. Use the auctionID as the key for the corresponding Auction object. In this assignment, you may provide your own implementation for the `AuctionTable` class, or you may use a hash table implementation provided by the Java API.

If you would like to know more about the Java API implementations, you should read the Orcale documentation for `java.util.Hashtable` and `java.util.HashMap`.

In addition to the standard hash table functionality, your class should include a function which will build an `AuctionTable` from a URL using the `BigData` library. This function should connect to the data source located at the URL, read in all the data members for each record stored in the data source, construct new `Auction` objects based on the data, and insert the objects in to the table. Exceptions should be thrown if the data source could not be connected to or contains invalid syntax structure.

- `public static AuctionTable buildFromURL(String URL) throws IllegalArgumentException`
  - ***Brief:***
    - **Uses the `BigData` library to construct an `AuctionTable` from a remote data source.**
  - ***Parameters:***
    - **`URL` - String representing the URL for the remote data source.**
  - ***Preconditions:***
    - **URL represents a data source which can be connected to using the `BigData` library.**
    - **The data source has proper syntax.**
  - ***Postconditions:***
    - **None.**
  - ***Returns:***
    - **The `AuctionTable` constructed from the remote data source.**
  - ***Throws:***
    - **`IllegalArgumentException`: Thrown if the URL does not represent a valid datasource (can't connect or invalid syntax).**
- `public void putAuction(String auctionID, Auction auction) throws IllegalArgumentException`
  - ***Brief:***
    - **Manually posts an auction, and add it into the table.**
  - ***Parameters:***
    - `auctionID - the unique key for this object`
    - `auction - The auction to insert into the table with the corresponding auctionID`
  - ***Preconditions:***
    - **None.**
  - ***Postconditions:***
    - **The item will be added to the table if all given parameters are correct.**
  - ***Throws:***
    - **`IllegalArgumentException`: If the given auctionID is already stored in the table.**
- `public Auction getAuction(String auctionID)`
  - ***Brief:***
    - **Get the information of an Auction that contains the given ID as key**
  - ***Parameters:***
    - `auctionID - the unique key for this object`
  - ***Returns:***
    - **An Auction object with the given key, null otherwise.**
- `public void letTimePass(int numHours) throws IllegalArgumentException`
  - ***Brief:***
    - **Simulates the passing of time. Decrease the timeRemaining of all Auction objects by the amount specified. The value cannot go below 0.**
  - ***Parameters:***
    - `numHours - the number of hours to decrease the timeRemaining value by.`

- ***Postconditions:***
  - **All Auctions in the table have their timeRemaining timer decreased. If the original value is less than the decreased value, set the value to 0.**
- ***Throws:***
  - **`IllegalArgumentException`: If the given numHours is non positive**

- **`public void removeExpiredAuctions()`**
  - ***Brief:***
    - **Iterates over all `Auction` objects in the table and removes them if they are expired (`timeRemaining == 0`).**
  - ***Postconditions:***
    - **Only open `Auction` remain in the table.**
- **`public void printTable()`**
  - ***Brief:***
    - **Prints the `AuctionTable` in tabular form.**

3. AuctionSystem

Write a fully-documented class named `AuctionSystem`. This class will allow the user to interact with the database by listing open auctions, make bids on open auctions, and create new auctions for different items. In addition, the class should provide the functionality to load a saved (serialized) `AuctionTable` or create a new one if a saved table does not exist.

On startup, the `AuctionSystem` should check to see if the file `auctions.obj` exists in the current directory. If it does, then the file should be loaded and deserialized into an `AuctionTable` for new auctions/bids (*See the section below for information on the `Serializable` interface*). If the file does not exist, an empty `AuctionTable` object should be created and used instead. Next, the user should be prompted to enter a username to access the system. This is the name that will be used to create new auctions and bid on the open auctions available in the table.

When the user enters 'Q' to quit the program, the auction table should be serialized to the file `auctions.obj`. That way, the next time the program is run, the auctions will remain in the database and allow different users to make bids on items. If you would like to 'reset' the auction table, simply delete the `auctions.obj` file.

- **An `AuctionTable` member variable:**
  - **`auctionTable`**
- **A `String` member variable:**
  - **`username`**
- **`public static void main(String[] args)`**
  - ***Brief:***
    - **The method should first prompt the user for a username. This should be stored in `username` The rest of the program will be executed on behalf of this user. Implement the following menu options:**
- **`(D) - Import Data from URL`**
- **`(A) - Create a New Auction`**
- **`(B) - Bid on an Item`**
- **`(I) - Get Info on Auction`**
- **`(P) - Print All Auctions`**
- **`(R) - Remove Expired Auctions`**
- **`(T) - Let Time Pass`**
- **`(Q) - Quit`**

Serializable Interface

You will also work with the idea of persistence. This means that our program should save all data from session to session. When we terminate a program, normally the data will be lost. We will preserve this data by using `Serializable` Java API and binary object files. All your classes should simply implement the java.io.Serializartion interface.

Example: **Your `AuctionTable` class contains information for all auctions saved in the electronic database. You would want to preserve this data, so you can load this data the next time you run your program. You would do the following:**

1. **Modify the AuctionTable so that it implements the `Serializable` interface. Also, the `Auction` class should also make this implementation. No other changes are necessary.**

```
public class AuctionTable implements Serializable
{
    // Member methods as is

}
```

2. **In your application that contains the hash, you can include code that will save the hash into a file so it can be read in again later. To do this, you need to create an `ObjectOutputStream` to send the data to, and then use the `writeObject` method to send the hash to the stream, which is stored in the specified file.**

```
FileOutputStream file = new FileOutputStream("auction.obj");
ObjectOutputStream outStream = new ObjectOutputStream(file);
AuctionTable auctions = new AuctionTable(/*Constructor Parameters*/);

// missing code here sets up the hash in some way
```

```
// the following line will save the object in the file
outStream.writeObject(auctions);
```

3. **When the same application (or another application) runs again, you can initialize the member using the serialized data saved from step 2 so you don't have to recreate the object from scratch. To do this, you need to create an `ObjectInputStream` to read the data from, and then use the `readObject` method to read the hash from the stream.**

```
FileInputStream file = new FileInputStream("auction.obj");
ObjectInputStream inStream = new ObjectInputStream(file);
AuctionTable auctions;

auctions = (AuctionLibrary) inStream.readObject();

// missing code here can use the same hash from step 2 now
```

Sample Input/Output:

// Comment in green, input in red, output in black

```
Starting...
No previous auction table detected.
Creating new table...                                   // Notify if loading or not.

Please select a username: student@stonybrook.edu

Menu:
     (D) - Import Data from URL
     (A) - Create a New Auction
     (B) - Bid on an Item
     (I) - Get Info on Auction
     (P) - Print All Auctions
     (R) - Remove Expired Auctions
     (T) - Let Time Pass
     (Q) - Quit


Please select an option: D
Please enter a URL: http://tinyurl.com/nbf5g2h
// TinyURL for http://www.cs.washington.edu/research/xmldatasets/data/auctions/ebay.xml

Loading...
Auction data loaded successfully!

// Menu not printed in sample i/o.

Please select an option: P

 Auction ID |      Bid    |         Seller          |           Buyer           |     Time    |  Item Info
// truncated to fit on one line
================================================================================================================
================================
  511601118 | $    620.00 | cubsfantony             |  gosha555@excite.com      | 110 hours | Pentium
III 933 System - 256MB PC133 SDram
  511448507 | $    620.00 | ct-inc                  |  petitjc@yahoo.com        |  54 hours | Pentium
III 800EB-MHz Coppermine CPU - 256
  511443245 | $ 1,025.00 | ct-inc                  |  hsclm9@peganet.com       |  54 hours | Intel
Pentium III 933EB-MHz Coppermine CPU
  511364992 | $    610.00 | bestbuys4systems        |  wizbang4                 |  53 hours | Genuine
Intel Pentium III 1000MHz Processo
  511357667 | $    535.00 | sales@ctgcom.com        |  chul2@mail.utexas.edu    |  53 hours | INTEL
Pentium III 800MHz - 256MB sdram -40


Please select an option: B
Please enter an Auction ID: 511364992

Auction 511364992 is OPEN
    Current Bid: $ 610.00

What would you like to bid?: 625.00
Bid accepted.
```

```
// Menu not printed in sample i/o.

Please select an option: P

 Auction ID |      Bid    |        Seller          |          Buyer          |    Time    |  Item Info
// truncated to fit on one line
================================================================================================
================================
  511601118 | $   620.00 | cubsfantony            |  gosha555@excite.com    | 110 hours | Pentium
III 933 System - 256MB PC133 SDram
  511448507 | $   620.00 | ct-inc                 |  petitjc@yahoo.com      |  54 hours | Pentium
III 800EB-MHz Coppermine CPU - 256
  511443245 | $ 1,025.00 | ct-inc                 |  hsclm9@peganet.com     |  54 hours | Intel
Pentium III 933EB-MHz Coppermine CPU
  511364992 | $   625.00 | bestbuys4systems       |  student@stonybrook.edu |  55 hours | Genuine
Intel Pentium III 1000MHz Processo
  511357667 | $   535.00 | sales@ctgcom.com       |  chul2@mail.utexas.edu  |  55 hours | INTEL
Pentium III 800MHz - 256MB sdram -40

// Menu not printed in sample i/o.

Please select an option: A

Creating new Auction as student@stonybrook.edu.
Please enter an Auction ID: 100000000
Please enter an Auction time (hours): 24
Please enter some Item Info: Core i5 2.7GHz - 4GB DDR3 - 750GB HDD

Auction 100000000 inserted into table.

// Menu not printed in sample i/o.

Please select an option: P

 Auction ID |      Bid    |        Seller          |          Buyer          |    Time    |  Item Info
// truncated to fit on one line
================================================================================================
================================
  511601118 | $   620.00 | cubsfantony            |  gosha555@excite.com    | 110 hours | Pentium
III 933 System - 256MB PC133 SDram
  511448507 | $   620.00 | ct-inc                 |  petitjc@yahoo.com      |  54 hours | Pentium
III 800EB-MHz Coppermine CPU - 256
  511443245 | $ 1,025.00 | ct-inc                 |  hsclm9@peganet.com     |  54 hours | Intel
Pentium III 933EB-MHz Coppermine CPU
  511364992 | $   625.00 | bestbuys4systems       |  student@stonybrook.edu |  55 hours | Genuine
Intel Pentium III 1000MHz Processo
  511357667 | $   535.00 | sales@ctgcom.com       |  chul2@mail.utexas.edu  |  55 hours | INTEL
Pentium III 800MHz - 256MB sdram -40
  100000000 |            | student@stonybrook.edu|                         |  24 hours | Core i5
2.7GHz - 4GB DDR3 - 750GB HDD

// Note that Bid and Buyer are missing for the new record. These correspond to 'null' attributes in
the database.

// Menu not printed in sample i/o.

Please select an option: Q

Writing Auction Table to file...                // Serialize the Auction Table.
Done!

Goodbye.

// Starting up again.
Starting...
Loading previous Auction Table...               // Loading from file.

Please select a username: buyer@gmail.com

// Menu not printed in sample i/o.

Please select an option: B
Please enter an Auction ID: 100000000
```

```
Auction 100000000 is OPEN
    Current Bid: None

What would you like to bid?: 1500.00
Bid accepted.

// Menu not printed in sample i/o.

Please select an option: P

 Auction ID |      Bid    |        Seller          |          Buyer          |    Time    |  Item Info
// truncated to fit on one line
================================================================================================
================================
  511601118 | $   620.00 | cubsfantony           |  gosha555@excite.com    | 110 hours | Pentium
III 933 System - 256MB PC133 SDram
  511448507 | $   620.00 | ct-inc                |  petitjc@yahoo.com      |  54 hours | Pentium
III 800EB-MHz Coppermine CPU - 256
  511443245 | $ 1,025.00 | ct-inc                |  hsclm9@peganet.com     |  54 hours | Intel
Pentium III 933EB-MHz Coppermine CPU
  511364992 | $   625.00 | bestbuys4systems      |  student@stonybrook.edu |  55 hours | Genuine
Intel Pentium III 1000MHz Processo
  511357667 | $   535.00 | sales@ctgcom.com      |  chul2@mail.utexas.edu  |  55 hours | INTEL
Pentium III 800MHz - 256MB sdram -40
  100000000 | $ 1,500.00 | student@stonybrook.edu|  buyer@gmail.com        |  24 hours | Core i5
2.7GHz - 4GB DDR3 - 750GB HDD

// Menu not printed in sample i/o.

Please select an option: T
How many hours should pass: 30

Time passing...
Auction times updated.

// Menu not printed in sample i/o.

Please select an option: P

 Auction ID |      Bid    |        Seller          |          Buyer          |    Time    |  Item Info
// truncated to fit on one line
================================================================================================
================================
  511601118 | $   620.00 | cubsfantony           |  gosha555@excite.com    |  80 hours | Pentium
III 933 System - 256MB PC133 SDram
  511448507 | $   620.00 | ct-inc                |  petitjc@yahoo.com      |  24 hours | Pentium
III 800EB-MHz Coppermine CPU - 256
  511443245 | $ 1,025.00 | ct-inc                |  hsclm9@peganet.com     |  24 hours | Intel
Pentium III 933EB-MHz Coppermine CPU
  511364992 | $   625.00 | bestbuys4systems      |  student@stonybrook.edu |  25 hours | Genuine
Intel Pentium III 1000MHz Processo
  511357667 | $   535.00 | sales@ctgcom.com      |  chul2@mail.utexas.edu  |  25 hours | INTEL
Pentium III 800MHz - 256MB sdram -40
  100000000 | $ 1,500.00 | student@stonybrook.edu|  buyer@gmail.com        |   0 hours | Core i5
2.7GHz - 4GB DDR3 - 750GB HDD

// Menu not printed in sample i/o.

Please select an option: B
Please enter an Auction ID: 100000000

Auction 100000000 is CLOSED
    Current Bid: $ 1,500.00

You can no longer bid on this item.

// Menu not printed in sample i/o.

Please select an option: I
Please enter an Auction ID: 511448507

Auction 51148507:
```

```
    Seller: ct-inc
    Buyer: petitjc@yahoo.com
    Time: 24 hours
    Info: Pentium III 800EB-MHz Coppermine CPU - 256MB PC133 SDRAM - 30.7GB IBM Deskstar ATA100
7200RPM

// Menu not printed in sample i/o.

Please select an option: R

Removing expired auctions...
All expired auctions removed.

// Menu not printed in sample i/o.

Please select an option: P

 Auction ID |      Bid    |        Seller          |          Buyer          |    Time    |  Item Info
// truncated to fit on one line
================================================================================================
================================
  511601118 | $    620.00 | cubsfantony            |  gosha555@excite.com    |  80 hours | Pentium
III 933 System - 256MB PC133 SDram
  511448507 | $    620.00 | ct-inc                 |  petitjc@yahoo.com      |  24 hours | Pentium
III 800EB-MHz Coppermine CPU - 256
  511443245 | $ 1,025.00 | ct-inc                  |  hsclm9@peganet.com     |  24 hours | Intel
Pentium III 933EB-MHz Coppermine CPU
  511364992 | $    625.00 | bestbuys4systems        |  student@stonybrook.edu |  25 hours | Genuine
Intel Pentium III 1000MHz Processo
  511357667 | $    535.00 | sales@ctgcom.com        |  chul2@mail.utexas.edu  |  25 hours | INTEL
Pentium III 800MHz - 256MB sdram -40

// Menu not printed in sample i/o.

Please select an option: Q

Writing Auction Table to file...                  // Serialize the Auction Table.
Done!

Goodbye.
```
