In this assignment, you will implement a database for baseball card collectors to keep track of their collections. A baseball card has a player name, team, position, card manufacturer, set, year, image size, price, statistics, and many other pieces of information. In this assignment, we will only store data about some of these. The database will hold at most 100 cards, a number which should be declared as a final variable.

1. Write a fully-documented class named BaseballCard which contains the player’s name (String), the card’s manufacturer (String), and year (int), price (double), and image size (int[], an integer array of size 2, representing the size along the x and y axes respectively). You should provide accessor and mutator methods for each variable. There should be separate accessors and mutators for the x and y components of the image size (i.e., you need getSizeX() and getSizeY()). The mutator method for the price variable should throw an exception if the new price is nonpositive. Also, include a constructor for the class, as well as clone() and equals() methods. The full list of required methods is:

- public BaseballCard() – constructor (you may include a constructor with parameters)
- Get/set name, manufacturer, year, price, sizeX, sizeY
- public Object clone() - The return value is a copy of this BaseballCard. Subsequent changes to the copy will not affect the original, nor vice versa. Note that the return value must be typecast to a BaseballCard before it can be used.
- public boolean equals(Object obj) - A return value of true indicates that obj refers to a BaseballCard object with the same attributes as this BaseballCard. Otherwise, the return value is false. Note: when comparing equality between the two cards’ image sizes, you cannot use the == operator on the size variables, because they are arrays. Using the == operator will simply check to see if the two variables refer to the same array, which does not take into consideration the fact that two different int arrays can actually represent the same list of ints.
2. Write a fully-documented class named CardCollection which stores an ordered list of BaseballCard objects in an array and provides an interface to interact with this list. Note that although arrays in Java are indexed starting with 0, the items in a CardCollection will be indexed starting with 1. A CardCollection can hold up to 100 cards at a time, so use the final variable MAX_CARDS = 100 (and refrain from writing the number 100 in your code). The class will be based on the following ADT specification:

```java
public class CardCollection

The CardCollection class implements an abstract data type for a list of baseball cards supporting some common operations on such lists.

- public CardCollection()
  Construct an instance of the CardCollection class with no BaseballCard objects in it.
  Postcondition:
  This CardCollection has been initialized to an empty list of BaseballCards.

- public int size()
  Determines the number of items currently in this collection.
  Preconditions:
  This CardCollection object has been instantiated.
  Returns:
  The number of BaseballCards in this CardCollection.

- public void addCard(BaseballCard newCard, int position)
  Parameters:
  newCard - the new BaseballCard object to add to this collection
  position - the position in this CardCollection where newCard will be inserted
  Preconditions:
  This CardCollection object has been instantiated and
  1 < position < items_currently_in_list + 1. The number of BaseballCard objects in this Menu is less than MAX_CARDS.
  Postcondition:
  The new BaseballCard is now stored at the desired position in the CardCollection. All cards that were originally in positions greater than or equal to position are moved back one position. (Ex: If there are 5 BaseballCards in a CardCollection, positions 1-5, and you
insert a new card at position 4, the new card will now be at position 4, the card that was at position 4 will be moved to position 5, and the card that was at position 5 will be moved to position 6).

Throws:
IllegalArgumentException
Indicates that position is not within the valid range.
FullCollectionException
Indicates that there is no more room inside of the CardCollection to store the new BaseballCard object.

Note 1:
position refers to the position in the CardCollection and not the position inside the array.

Note 2:
Inserting an item to position (items_currently_in_list + 1) is effectively the same as adding a card to the end of the collection.

• public void addCard(BaseballCard newCard)
Works just like addCard(BaseballCard newCard, int position), except adds to the end of the list. Note: this method can be written in one line using the addCard() method above and the size() method above.

• public void removeCard(int position)
Parameters:
position - the position in the CardCollection where the BaseballCard will be removed from.
Preconditions:
This CardCollection object has been instantiated and 1 < position < items_currently_in_list.
Postcondition:
The card at the desired position in the collection has been removed. All cards that were originally in positions greater than or equal to position are moved forward one position. (Ex: If there are 5 items in a collection, positions 1-5, and you remove the item at position 4, the item that was at position 5 will be moved to position 4).
Throws:
IllegalArgumentException
Indicates that position is not within the valid range.
Note:
position refers to the position in the CardCollection and not the position inside the array.

• public BaseballCard getCard(int position)
Get the BaseballCard at the given position in
this CardCollection object.

Parameters:
position - position of the card to retrieve

Preconditions:
This CardCollection object has been instantiated and
1 ≤ position ≤ items_currently_in_list.

Returns:
The card at the specified position in this Menu object.

Throws:
IllegalArgumentException
Indicates that position is not within the valid range.

Note:
position refers to the position in the CardCollection and not the
position inside the array.

• public void trade(CardCollection other, int myPosition, int theirPosition)
Exchange a card from this collection for a card from another
collection.

Parameters:
other – the CardCollection we will be trading with
myPosition – the position within this collection of the card
to trade
theirPosition – the position within the other collection of
the card to trade

Preconditions:
Both CardCollection objects have been instantiated and
1 ≤ myPosition ≤ items_currently_in_this_list and
1 ≤ theirPosition ≤ items_currently_in_other_list.

Throws:
IllegalArgumentException
Indicates that either position is not within the valid range.

Note:
position refers to the position in the CardCollection and not the
position inside the array.

• public void exists(BaseballCard card)
Check whether this collection contains the given card

Parameters:
card – the BaseballCard we are looking for

Preconditions:
This CardCollection and the BaseballCard object have both been
instantiated

Throws:

Returns:

True, if this CardCollection contains the card, false otherwise.

IllegalArgumentException

Indicates that either position is not within the valid range.

• public void printAllCards()

Prints a neatly formatted table of each item in the collection on its
own line with its position number as shown in the sample output.

Preconditions:

This CardCollection object has been instantiated.

Postcondition:

A neatly formatted table of each card in the collection on its own line
with its position number has been displayed to the user.

Hint:

If your toString() method is implemented correctly as described
below, you will simply need to call it and print the results to the user.

• public String toString()

Gets the String representation of this CardCollection object, which is
a neatly formatted table of each BaseballCard in
the CardCollection on its own line with its position number as shown
in the sample output.

Returns:

The String representation of this CardCollection object.

3. Write a fully documented class named CollectionManager that is
based on the following specification:

public class CollectionManager

The CollectionManager Java application tests the data structure
classes designed above and the operations defined on them.

• main

    public static void main(String[] args)

    The main method runs a menu driven application which first
creates two empty CardCollections (referred to by the user as
A and B), and then prompts the user for a command
selecting the operation. Once an operation is selected, the
program prompts for any additional information required to
perform the operation, and then actually performs the
operation. The operations, their action letter, and additional
information required are listed below.
• Add Card: A <Collection to add to> <Position> <Name> <Manufacturer> <Year> <Price> <Size>
  Construct and add the card to the indicated collection at the indicated position.
• Get Card: G <Collection> <Position>
  Print out the name, manufacturer, year, and price of the card at the specified position in the indicated collection.
• Remove Card: R <Collection> <Position>
  Remove the card at the specified position in the indicated collection.
• Print All Cards: P
  Print the list of all cards in each collection.
• Look for Card: L <Name> <Manufacturer> <Year> <Price> <Size>
  Construct the card from the input, and print out whether or not it exists in either collection. You must use the Card's equals() method for comparison.
• Size: S
  Print the number of cards in each collection.
• Value: V
  Print the total value of each collection.
• Update name: N <Collection> <Position> <New name>
  Update the name of the card in the indicated position in the indicated collection.
• Update price: E <Collection> <Position> <New price>
  Update the price of the card in the indicated position in the indicated collection.
• Copy card: C <From collection> <Position> <To collection>
  Copy the card at the specified position in <From collection> to the end of <To collection>, using the card's clone() method. Note: <From collection> and <To collection> may be the same.
• Trade: T <Position A> <Position B>
  Exchange the card at position A in collection A with the card at position B in collection B.
• Quit: Q
  Terminate the program gracefully.

4. You will also need classes to handle the exceptions thrown (see class specifications above for the exception classes you need).

Note: You may include additional methods in any class as necessary or as you find convenient.

INPUT FORMAT:

• Each menu operation is entered on its own line and should be case insensitive (i.e. 'q' and 'Q' are the same).
• Check to make sure that the position, if required, is valid. If not, print an error message and return to the menu.
• For the Add Card command, if the input information is valid, construct the object accordingly. Otherwise, print an error message and return to the menu.
• You may assume that the lengths of the input for the player names and card manufacturers are at most 25 characters long

OUTPUT FORMAT:

• Each command should output the result (as shown in sample IO below) after each operation is performed.
• All menu operations must be accompanied by a message indicating what operation was performed and whether or not it was successful.
• All lists must be printed in a nice and tabular form as shown in the sample output. You may use C style formatting as shown in the following example. The example below shows two different ways of displaying the name and address at pre-specified positions 21, 26, 19, and 6 spaces wide. If the '-' flag is given, then it will be left-justified (padding will be on the right), else the region is right-justified. The 's' identifier is for strings, the 'd' identifier is for integers . Giving the additional '0' flag pads an integer with additional zeroes in front.
•
  String name = "Doe Jane";
  String address = "32 Bayview Dr.";
  String city = "Fishers Island, NY";
  int zip = 6390;
•
  System.out.println(String.format("%-21s%-26s%19s%06d", name, address, city, zip));
  System.out.printf("%-21s%-26s%19s%06d", name, address, city, zip);
•
  Doe Jane 32 Bayview Dr. Fishers Island, NY 06390
  Doe Jane 32 Bayview Dr. Fishers Island, NY 06390

HINTS:

• Remember that the position parameter to all of the methods listed in the CardCollection class refers to the position of a card within the collection (starting at position 1) and not the position inside of the array (which starts at position 0). There are two ways that you can handle this issue:
  o Store item 1 in array position 0, item 2 in array position 1, and so on and so forth. Inside each method, subtract one from the
position given by the parameter to find the appropriate position within the array.

- Define your array such that it is of size MAX_CARDS + 1 instead of MAX_CARDS. Store item 1 in array position 1, item 2 in array position 2, and so on and so forth. Position 0 of the array will not be used.

**EXTRA CREDIT:**

**SAMPLE INPUT/OUTPUT:**

Output shown in black. **User input shown in red. Comments shown in green.**

Main menu:

A) Add Card  
C) Copy  
E) Update price  
G) Get Card  
L) Locate Card  
N) Update name  
P) Print All Cards  
R) Remove Card  
S) Size  
T) Trade  
Q) Quit

Select an operation: A

Enter the collection: A  
Enter the name: Johnny Damon  
Enter the manufacturer: Topps  
Enter the year: 2004  
Enter the size: 100 300  
Enter the price: 9.50  
Enter the position: 1

Added Johnny Damon, Topps 2004, 100x300, $9.50 at position 1 of collection A

// Menu not shown in sample input/output

Select an operation: A

Enter the collection: B  
Enter the name: Derek Jeter  
Enter the manufacturer: Score  
Enter the year: 2007  
Enter the size: 200 300  
Enter the price: 7.50
Enter the position: 1

Added Derek Jeter, Score 2007, 200x300, $7.50 at position 1 of collection B

// Menu not shown in sample input/output

Select an operation: A

Enter the collection: A
Enter the name: David Ortiz
Enter the manufacturer: Upper Deck
Enter the year: 2010
Enter the size: 200 400
Enter the price: 10.50
Enter the position: 1

Added David Ortiz, Upper Deck 2010, 200x400, $10.50 at position 1 of collection A // Pushes the previous card up to position 2

// Menu not shown in sample input/output

Select an operation: P

Collection A:

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Manufacturer</th>
<th>Year</th>
<th>Price</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>David Ortiz</td>
<td>Upper Deck</td>
<td>2010</td>
<td>$10.50</td>
<td>200x400</td>
</tr>
<tr>
<td>2</td>
<td>Johnny Damon</td>
<td>Topps</td>
<td>2004</td>
<td>$9.50</td>
<td>100x300</td>
</tr>
</tbody>
</table>

Collection B:

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Manufacturer</th>
<th>Year</th>
<th>Price</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Derek Jeter</td>
<td>Score</td>
<td>2007</td>
<td>$7.50</td>
<td>200x300</td>
</tr>
</tbody>
</table>

// Menu not shown in sample input/output

Select an operation: S

Collection A has 2 cards. Collection B has 1 card.

// Menu not shown in sample input/output

Select an operation: C

Enter the collection to copy from: A
Enter the position of the card to copy: 2
Enter the collection to copy to: A

Copied Johnny Damon, Topps 2004, 100x300, $9.50 into position 3 of collection A

// Menu not shown in sample input/output

Select an operation: N
Enter the collection: A
Enter the position: 3
Enter the new name: Dustin Pedroia

Changed name of collection A position 3 from “Johnny Damon” to “Dustin Pedroia”

// Menu not shown in sample input/output

Select an operation: E

Enter the collection: A
Enter the position: 3
Enter the new price: 6

// Menu not shown in sample input/output

Select an operation: P

Collection A:

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Manufacturer</th>
<th>Year</th>
<th>Price</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>David Ortiz</td>
<td>Upper Deck</td>
<td>2010</td>
<td>$10.50</td>
<td>200x400</td>
</tr>
<tr>
<td>2</td>
<td>Johnny Damon</td>
<td>Topps</td>
<td>2004</td>
<td>$9.50</td>
<td>100x300</td>
</tr>
<tr>
<td>3</td>
<td>Dustin Pedroia</td>
<td>Topps</td>
<td>2004</td>
<td>$6.00</td>
<td>100x300</td>
</tr>
</tbody>
</table>

Collection B:

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Manufacturer</th>
<th>Year</th>
<th>Price</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Derek Jeter</td>
<td>Score</td>
<td>2007</td>
<td>$7.50</td>
<td>200x300</td>
</tr>
</tbody>
</table>

// Menu not shown in sample input/output

Select an operation: T

Enter the position of the card to trade from collection A: 3
Enter the position of the card to trade from collection B: 1

Traded Derek Jeter, Score 2007, 200x300, $7.50 for Dustin Pedroia, Topps 2004, 100x300, $6.00

// Menu not shown in sample input/output

Select an operation: R

Enter the collection to remove from: A
Enter the position to remove: 2

Removed Johnny Damon, Topps 2004, 100x300, $9.50 from collection A

// Positions 3+ move up 1 position

// Menu not shown in sample input/output
Select an operation: P

Collection A:

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Manufacturer</th>
<th>Year</th>
<th>Price</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>David Ortiz</td>
<td>Upper Deck</td>
<td>2010</td>
<td>$10.50</td>
<td>200x400</td>
</tr>
<tr>
<td>2</td>
<td>Derek Jeter</td>
<td>Score</td>
<td>2007</td>
<td>$7.50</td>
<td>200x300</td>
</tr>
</tbody>
</table>

Collection B:

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Manufacturer</th>
<th>Year</th>
<th>Price</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Dustin Pedroia</td>
<td>Topps</td>
<td>2004</td>
<td>$6.00</td>
<td>100x300</td>
</tr>
</tbody>
</table>

Select an operation: V

The total value of collection A is $18.00. The total value of collection B is $6.00.

Select an operation: L

Enter the name: David Ortiz
Enter the manufacturer: Upper Deck
Enter the year: 2010
Enter the size: 200 400
Enter the price: 10.50

The card is in collection A. The card is not in collection B.

Select an operation: Q

Quitting.
Homework 1 - GRADING KEY

1. (15 points) Documentation and coding style:
   - Precise and correct specifications [5 points]
   - Indentation [3 points]
   - Variable names [2 points]
   - Javadoc Style *(see NOTE). [5 points]

   NOTE: You only need to submit your .java files. We will use the javadoc *.java command to create the javadoc files.

2. (15 points) Source code:
   - Implementation done as specified (additional methods are OK) [10 points]
   - Throw an exception back to the calling method(s) if a precondition is violated. [5 points]

3. (65 points) Program correctness:
   - Compiles without error (no partial credit) [12 points]
   - Clone and equals work correctly [4 points]
   - Menu operations work correctly:
     - A add card [5 points]
     - G get card [3 points]
     - R remove card [5 points]
     - P print all cards [5 points]
     - L look for card [4 points]
     - S size [4 points]
     - V value [4 points]
     - N update name [4 points]
     - P update price [4 points]
     - C copy card [5 points]
     - T trade [5 points]
     - Q quit [1 point]

4. (5 points) Program efficiency: size() and getItem() should be O(1) and all other methods should be O(n)

5. Extra credit:
   - Use GUI for all the user interface (input/output). [up to 8 points depending on how well you incorporate the GUI options]
   - The best program in class receives up to 6 extra points and will be posted as the sample answer. (The author may remove his/her name before posting the program)