PROBLEM 1 : (*Mystery Repeat Repeat Repeat:* (24 pts))

PART A *(10 pts)*:

Consider the following Mystery method. Note the precondition that there must be at least two blanks in *phrase*.

```
// precondition: assume phrase has at least two blanks.
public String Mystery (String phrase)
{
   int pos = phrase.indexOf(" ");
   int pos2 = phrase.indexOf(" ",pos+1);

   return phrase.substring(pos2+1) + ", " + phrase.substring(0,pos);
}
```

A. What type is the return value for the method Mystery?

B. List all the local variables and their types in Mystery.

C. What is the return value for the call Mystery("Richard H. Brodhead")?

D. What is the return value for the call Mystery("A B C D E")?

E. Describe in words what the method Mystery does.

PART B *(14 pts)*: Consider the following Mystery2 method.

```
public int Mystery2(ArrayList<Integer> numbers)
{
   int count = 0;
   for (Integer num: numbers)
   {
      if (num % 2 == 0)
      {
         count += num;
      }
   }
   return count;
}
```

A. How many parameters does Mystery2 have? List their names.

B. What type of values are stored in the ArrayList?

C. Suppose *values* is an ArrayList<Integer> and has the values 5, 8, 10, 17, and 3 stored in this order from position 0 to position 4. What is the return value of the call Mystery2(values)?

D. Describe in words what the method Mystery2 does.

E. Rewrite the complete body of Mystery2 using a for loop that is not a collections for loop.

F. Rewrite the complete body of Mystery2 using a while loop instead of a for loop.

PROBLEM 2 : (*Read a good book lately?:* (12 pts))

Consider the following two classes.

```
public class Book {

   private String myTitle;
   private String myAuthor;
   private int myPages;

   public Book(String Title, String author, int pages)
   {  myTitle = Title;
      myAuthor = author;
      myPages = pages;     }

   public String getTitle()
   {   return myTitle;     }

   public String getAuthor()
   {   return myAuthor;    }

   public int getPages()
   {   return myPages;     }
}

public class PictureBook extends Book {

   private String myIllustrator;

   public PictureBook(String title, String author,
         String illustrator, int pages)
   {
      super(title, author, pages);
      myIllustrator = illustrator;
   }

   public String getAuthor()
   {   return super.getAuthor() + " " + myIllustrator;
   }

   public void setAuthor(String name)
   {   myAuthor = name;
   }
```

```
}
```

A : Which of the two classes is the superclass?

B : Eclipse would flag one error in the PictureBook class. Explain which line causes the error and how to fix it.

C : Assume the error in the PictureBook class from part B is fixed. Consider the following 4 sections of code. State if the two lines of code are valid or contain an error. If they contain an error, then state what the error is. If they do not contain an error, then list the corresponding output.

```
// 1.
   Book book1 = new Book(‘‘Cross’’, ‘‘Patterson’’, 400);
   System.out.println(book1.getAuthor());



// 2.
   Book book2 = new Book(‘‘Eragon’’, ‘‘Paolini’’);
   System.out.println(book2.getAuthor());


// 3.
   Book book3 = new PictureBook(‘‘Goodnight Moon’’, ‘‘Brown’’, ‘‘Hurd’’, 32);
   System.out.println(book3.getAuthor());



// 4.
   PictureBook book4 = new Book(‘‘Polar Express’’, ‘‘Van Allsburg’’,32);
   System.out.println(book4.getAuthor());
```

PROBLEM 3 : (*Who is hungry?* (52 pts))

Consider the class GroceryStore shown below. It stores information about all the items it has including the name, price and the quantity of the item it has in stock. This information is stored in ArrayLists in which the kth item in each ArrayList corresponds.

```
public class GroceryStore {

   private ArrayList<String> myItems;       // name of items
   private ArrayList<Double> myPrices;      // price of items
   private ArrayList<Integer> myQuantity;   // quantity of items in stock

   // Constructor, assume no duplicate items in the file
```

```
   public GroceryStore (Scanner input)
   {   // code not shown    }

   // returns number of different types of items
   public int getNumberOfItems()
   {   // code not shown       }

   // returns total number of items in store
   public int getTotalNumberOfItems()
   {   // code not shown       }

   // returns price of specific item, returns -1 if item not in store
   public double getPriceOfitem(String item)
   {  // code not shown        }

   // returns number of unique items with price between minPrice and maxPrice, inclusive
   public int getNumberOfItemsWithPrice(double minPrice, double maxPrice)
   {  // code not shown   }

   // add an item to the grocery store if it is not there. If it is
   // already there, update the quantity and use this price as the new price.
   public void addItem(String item, double price, int quantity)
   {    // code not shown  }

   // Given a string of items, returns an ArrayList of the items
   public ArrayList<String> ItemsInCart (String cart)
   { // code not shown  }

   //returns the cost of all the items in the basket
   public double RingUpBasket(String cart)
   {  // code not shown  }
}
```

PART A ***(40 pts)***:

PART A1 ***(6 pts)***: Consider the constructor for the class GroceryStore. It should read in the stores' inventory from a file and store it in the three ArrayLists myItems, myPrices and myQuantity, such that the kth item in each ArrayList corresponds to the kth item in the other ArrayLists.

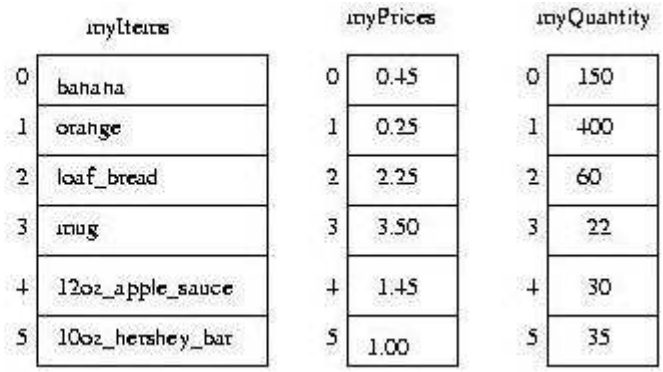
Note that the Scanner input is already bound to a file. The file is in the following format. Each line in the file has the name of an item (containing no blanks), the price of the item and the quantity of the item in stock.

Consider the input file shown on the left and the corresponding ArrayLists shown on the right.

Complete the constructor for the GroceryStore class below.

```
   // assume no duplicate items in the file
```

```
banana 0.45 150
orange 0.25 400
loaf_bread 2.25 60
mug 3.50 22
12oz_apple_sauce  1.45 30
10oz_hershey_bar 1.00 35
```




```
public GroceryStore (Scanner input)
{

}
```

PART A2 ***(3 pts)***:

Complete the method getNumberOfItems that returns the number of unique items in the grocery store. Assume that the entries in the ArrayList myItems are all unique.

For the example in PART A, getNumberOfItems() would return 6 as there are six items in the ArrayLists.

```
// returns number of different types of items
public int getNumberOfItems()
{

}
```

PART A3 ***(4 pts)***:

Complete the method getTotalNumberOfItems() that returns the total number of items in the store.

For the example in PART A, getTotalNumberOfItems() would return 697, which is the sum of all the quantities of the items in stock.

```
// returns total number of items in store
public int getTotalNumberOfItems()
{

}
```

PART A4 ***(5 pts)***:

Complete the method getPriceOfItem(String item) that returns the price of *item*.

For the example in PART A, getPriceOfItem(mug) would return 3.50. The call getPriceOf-Item(rootbeer) would return -1 as rootbeer is not an item at the store.

```
// returns price of specific item, returns -1 if item not in store
public double getPriceOfitem(String item)
{

}
```

PART A5 ***(5 pts)***:

Complete the method getNumberOfItemsWithPrice(double minPrice, double maxPrice) that returns the number of unique items in the inventory whose price is between minPrice and maxPrice, inclusive. For the example in PART A, getNumberOfItemsWithPrice(1.00, 2.00) would return 2, as there are two items with prices 1.00 and 1.45.

```
public int getNumberOfItemsWithPrice(double minPrice, double maxPrice)
{

}
```

PART A6 ***(5 pts)***:

Complete the method addItem(String item, double price, int quantity) that adds an item to the grocery store if it is not there. If the item is already there, it replaces the price with this new price and updates the quantity by adding quantity to the previous value.

For the example in PART A, addItem(mug, 4.00, 20) would update the mug in the grocery store inventory to price 4.00 and quantity 42. The call addItem(8oz_mustard, 1.34, 18) adds a new entry to the inventory.

```
public void addItem(String item, double price, int quantity)
{

}
```

PART A7 ***(6 pts)***:

Complete the method ItemsInCart(String cart), that is given a string of the names of items in a grocery cart, and returns an ArrayList of these names. Assume the items are separated by one blank.

For the example in PART A, ItemsInCart("banana banana loaf_bread scotch_tape") would return an ArrayList with four entries: banana, banana, loaf_bread, scotch_tape.

```
public ArrayList<String> ItemsInCart (String cart)
{

}
```

PART A8 *(6 pts)*:

Complete the method RingUpCart(String cart), that is given a string of the names of items in a grocery cart, and returns the price of the items in the cart. If an item in the cart is not in the inventory, then print an error message for that item and do not include its price in the total. Assume the items in cart are separated by one blank.

You must call the method ItemsInCart that you wrote in part F. Assume ItemsInCart works correctly, regardless of what you wrote for part F.

For the example in PART A, RingUpCart("banana banana loaf_bread scotch_tape") would print an error message for scotch_tape and return 3.15, the cost of the two bananas (0.45 each) and 2.25 for the loaf_bread.

```
// returns the cost of all the valid items in the basket,
// assume there is one blank between each pair of items in cart,
// prints an error message for items not in inventory
public double RingUpBasket(String cart)
  {



   }
```

PART B *(12 pts)*:

The class `GroceryStore` could be redesigned by adding a second class called `GroceryItem` to represent an item in stock.

```
public class GroceryItem {

   String myItem;    // name of item
   double myPrice;   // price of item
   int myQuantity;   // quantity of item

   // constructor
   public GroceryItem (String item, double price, int quantity)
   {
      myItem = item;
      myPrice = price;
      myQuantity = quantity;
   }

   // returns name of item
   public String getItem()
   {  // code not shown
   }
```

```
   // returns price of item
   public double getPrice()
   {  // code not shown
   }

   // returns quantity of item
   public int getQuantity()
   {  // code not shown
   }

   // changes price of item to price
   public void setPrice(double price)
   {   // code not shown
   }

   // changes quantity of item to quantity
   public void setQuantity(int quantity)
   {  // code not shown
   }
}
```

PART B1 ***(3 pts)***:

Give the new state for the GroceryStore class that uses the GroceryItem class.

PART B2 ***(5 pts)***:

Write the new constructor for the GroceryStore class. Note that the Scanner input is already bound to a file. The file is in the same format as before. Each line in the file has the name of an item (containing no blanks), the price of the item and the quantity of the item in stock.

```
   // assume no duplicate items in the file
   public GroceryStore (Scanner input)
   {









}
```

PART B3 ***(4 pts)***:

With the changes made to the GroceryStore class in Part B, complete the method getPrice-OfItem(String item) that returns the price of *item*.

```
   // returns price of specific item, returns -1 if item not in store
   public double getPriceOfitem(String item)
   {

   }
```

```
public class String
{
    // Returns the length of this string.
    public int length ()

    // Returns a substring of this string that begins at the specified
    // beginIndex and extends to the character at index endIndex - 1.
    public String substring (int beginIndex, int endIndex)

    // Returns a substring of this string that begins at the specified
    // beginIndex and extends to the end of the string.
    public String substring (int beginIndex)

    // Returns position of the first occurrence of str, returns -1 if not found
    public int indexOf (String str)

    // Returns the position of the first occurrence of str after index start
    //  returns -1 if str is not found
    public int indexOf (String str, int start)

    // returns character at position index
    public char charAt(int index)

    // returns true if str has the exact same characters in the same order
    public boolean equals(String str)

    // returns the string as an array of characters
    public char [] toCharArray()
}

public class ArrayList
{
    // Constructs an empty list
    public ArrayList ()

    // Returns the number of elements in this list.
    public int size ()
```

```
    // Returns element at position index in this list.
    public Object get (int index)

    // Replaces the item at position index with element.
    public Object set (int index, Object element)

    // Appends specified element to end of this list.
    public boolean add (Object o)
}


public class Scanner
{
    // Create Scanner that reads data from a file.
    public Scanner (File file)

    // Create Scanner that reads data from a string.
    public Scanner (String str)

    // Change delimiters used to separate items
    public void useDelimiter (String characters)

    // Check if more items are available
    public boolean hasNext ()

    // Get next delimited item as a string
    public String next ()

    // Get next line as a string
    public String nextLine ()

    // Get next delimited item as an integer value
    public int nextInt ()

    // Get next delimited item as a Double value
    public int nextDouble ()
}
```