Aaron Jones
5/16/14
Lab 10: Data Abstraction

```
//  Tester class.  Do not change anything in this class.

/*  Tasks for the data abstraction section:
 *   - Add code to the 'Transaction' class to implement the Comparable interface
 *   - Complete the InvoiceItem and Invoice classes
 *      (See comments in the InvoiceItem.java and Invoice.java)
 *  See also the Item.java class and expected output in output.txt
 *  Note your output will vary slightly for the data stamp - hh:mm:ss
 */
import java.util.*;
public class Tester
{
   public static void main(String[] args)
   {
      Transaction[] trans = new Transaction[4];
      PO po;
      Invoice iv;

      po = new PO(101, 502, "Pending");
      po.addItem(42, "Lumia 900", 425.00, 2, 1.50);
      trans[3] = po;

      po = new PO(102, 501, "Approved");
      po.addItem(24, "Lumia 300", 433.00, 2, 2.0);
      po.addItem(86, "iPhone 4S", 721.50, 1, 1.75);
      trans[2] = po;

      iv = new Invoice(201, 902, false);
      iv.addItem(1255, "Samsung Flash", 125.00, 2, 5);
      iv.addItem(198, "HTC m7", 533.00, 1, 4);
      trans[1] = iv;

      iv = new Invoice(202, 901, true);
      iv.addItem(681, "Lumia 822", 470.50, 3, 4);
      iv.addItem(199, "HTC One", 389.00, 1, 5);
      iv.addItem(1255, "Samsung Flash", 125.00, 2, 3);
      trans[0] = iv;

      //  This call to Arrays.sort will require implementation of the
      //  Comparable interface in the Transaction class...
      Arrays.sort(trans);

      for (Transaction t : trans)
      {
```

Aaron Jones
5/16/14
Lab 10: Data Abstraction

```
        //  Show formatted PO's and invoices...
        System.out.println(t);

        //  Process inventory...
        for (Item i : t.getItems())
        {
          System.out.println("\t" + i.changeInventory());
        }
      }
  }
}

import java.util.*;
/*  Modify this source code in the space indicated only!
 *
 *  Implement the Comparable interface in this class (see below)...
 */


public abstract class Transaction implements Comparable<Transaction>          //
<--  Your code goes here
{
  // Note the tranID variable is private - do not change
  private   int  tranID;
  protected int  participantID;
  protected Date tranDate;
  protected ArrayList<Item> items;

  public Transaction(int tranID, int participantID)
  {
    this.tranID = tranID;
    this.participantID = participantID;
    this.tranDate = Calendar.getInstance().getTime();
    items = new ArrayList<Item>();
  }

  protected final ArrayList<Item> getItems()
  {
    return this.items;
  }

  /*  Implement the Comparable interface here.
   *  Sort first by participantID and if the participantID's are the
   *  same, sort by tranID.
```

Aaron Jones
5/16/14
Lab 10: Data Abstraction

```
   */

   //  Your code goes here...

   @Override
   public int compareTo(Transaction that)
   {
     if(this.participantID > that.participantID)
        return 1;
     else if(participantID < that.participantID)
        return -1;
     else if(this.participantID == that.participantID)
        {
          if(this.transID > that.transID)
             return 1;
          else if(transID < that.transID)
             return -1;
        }
     else
        return 0;
   }




  @Override
  public String toString()
  {
    return tranID + ", Date: " + tranDate;
  }

}

import java.util.*;
/*  Modify this source code in the space indicated only!
 *
 *  Implement the Comparable interface in this class (see below)...
 */


public abstract class Transaction implements Comparable<Transaction>          //
<--  Your code goes here
{
```

Aaron Jones
5/16/14
Lab 10: Data Abstraction

```
// Note the tranID variable is private - do not change
private   int  tranID;
protected int  participantID;
protected Date tranDate;
protected ArrayList<Item> items;

public Transaction(int tranID, int participantID)
{
  this.tranID = tranID;
  this.participantID = participantID;
  this.tranDate = Calendar.getInstance().getTime();
  items = new ArrayList<Item>();
}

protected final ArrayList<Item> getItems()
{
  return this.items;
}

/*  Implement the Comparable interface here.
 *  Sort first by participantID and if the participantID's are the
 *  same, sort by tranID.
 */

 //  Your code goes here...

 @Override
 public int compareTo(Transaction that)
 {
  if(this.participantID > that.participantID)
     return 1;
  else if(participantID < that.participantID)
     return -1;
  else if(this.participantID == that.participantID)
     {
       if(this.transID > that.transID)
          return 1;
       else if(transID < that.transID)
          return -1;
     }
  else
     return 0;
 }
```

Aaron Jones
5/16/14
Lab 10: Data Abstraction

```
  @Override
  public String toString()
  {
    return tranID + ", Date: " + tranDate;
  }

}


//  Item class - do not change anything in this class

public abstract class Item
{
  protected int itemID;
  protected String description;
  protected double unitPrice;
  protected int quantity;

  public Item(int itemID, String description, double unitPrice, int quantity)
  {
    this.itemID = itemID;
    this.description = description;
    this.unitPrice = unitPrice;
   this.quantity = quantity;
  }

  protected abstract String changeInventory();

  @Override
  public String toString()
  {
    return "\tItem: " + itemID + " - " + description +
         " Quantity: " + quantity + ", Cost: " + unitPrice * quantity;
  }

}

  import java.util.*;

  public class Invoice extends Transaction
```

Aaron Jones
5/16/14
Lab 10: Data Abstraction

```
{
//  Add a true/false variable - 'isPaid'...
    protected boolean isPaid;


//  Add an Explicit Value Constructor.
//   Parameters: int tranID, int customer, boolean paid
//   Note 'tranID' is private to the Item class
    public Invoice(int tranID, int customer, boolean paid)
    {
      super.tranID = tranID;
      super.participateID = customer;
      this.isPaid = paid;

    }




//  Add an 'addItem' method here.
//   The method creates a new InvoiceItem object and adds it to the ArrayList
//    inherited from the Transaction class.
//   Parameters: int itemID, String desc, double price, int qty, int ounces

   protected ArrayList<List> addItem(InvoiceItem o)
   {
     super.items.add(o);
     return items;
   }




//  Override toString here.
//   Return value must include 'tranID' from Transaction, 'isPaid'
//   and listing of InvoiceItems...
//   See example output in output.txt.

   @Override
   public String toString()
   {
     return "Customer: " + tranID + "Paid: " + isPaid;
```

```
    }




  }


  public class InvoiceItem extends Item
  {

  //  Add variable 'ouncesPerUnit' as an int...
      protected int ouncesPerUnit;



  //  Add constructor.
  //   Parameters are: int itemID, String description, double price, int
quantityOrdered, int ouncesPerUnit
     public InvoiceItem(int itemID, String description, double price, int
quantityOrdered, int ouncesPerUnit)
    {
       super.itemID = itemID;
      super.description = description;
       super.price = price;
       super.quantityOrdered = quantityOrdered;
       this.ouncesPerUnit = ouncesPerUnit;
    }







  //  Add 'changeInventory' method...
  //   The method creates a String that reports the processing that would take
place.
  //   See example output in output.txt

  @Override
  protected String changeInventory()
  {
```

Aaron Jones
5/16/14
Lab 10: Data Abstraction

```
      return "Inventory has been relieved for" + this.description + " by " +
this.quantityOrdered;
   }



   //  Override 'toString' method here.
   //   Return 'toString' from Item class plus the ounces per unit...
   @Override
   public String toString()
   {
      return super.toString() + "Ounces: " + ouncesPerUnit + "\n";
   }




   }
```