CS 1331 Homework 7

Due Thursday, February 28th , 2013 8:00pm

IMPORTANT NOTICE

We have been seeing a lot of general problems with homework submissions so far. We want you all to be 100% clear on the rules.

- Safe Submission
  - make sure that you are turning in what you mean to!!
  - any submission that **does not compile and run** is a **0** on that program
  - any submission that **lacks .java source files (.class files are not okay)** is a **0** on the program as well – we need your source code
  - how to double check? After uploading your homework, download the turned in files from T-Square and save them in a new folder. Go to that folder, look at the file names, compile, and run/test. Pay attention also to the email from T-Square.
- Do your own work
  - You are welcome to help each other, work through conceptual problems, and even some weird code issues, but **NEVER** give your code to anyone else
  - Turning in someone else's code is plagiarism and will not be tolerated.
- All instance variables **MUST be private**
- **Style matters.** Indentation, good variable names, reasonable line length (keep it to around 80 characters per line)
- Do not overuse static
  - In this program, the only thing that should be static is your main method and the static constants in Item and it's subclasses
  - using static to try to make things compile is not the way to go
- Declare all of your instance variables before your constructor, but do not initialize any of them there – all initialization should be done in the constructor of the class

Introduction

This week the homework is meant to give you some hands on experience with inheritance and a little bit of polymorphism, as well as some more practice with arrays.

We will be making a text-based interactive storefront where a Customer can buy various items that the store has in stock. We will do this by making many different classes that interact with one another. We will have an Item class, and two child classes of Item. We will also have a ShoppingCart, Customer, Store, and StoreDriver.

Also make sure to javadoc all your files.

If you get stuck, feel free to post on Piazza or come to office hours in CCB107! CCB 107 is a large room with comfy tables. The hours are posted on T-Square.

Also, for each of your classes, the class outline is there to help you out. If you follow the outline we give you, your program will follow good OOP style, but you are free to add in things or make things different, as long as you provide all of the required functionality and still follow good OOP style. Don't get adventurous if you are not quite sure what you are doing!

7.1 Item.java

In order to have a Store, we first need to have a way to represent things to buy. An Item should have a name, a price, and a tax rate. It should also have a way to get the price of the item, get the price after tax of the item, and get a nicely formatted String representation of the item.

Item should also have a constant for the tax rate on Items – all Items (that are not actually Grocery and not Software items) should have the same tax. Use a value of 0.09 for this standard tax rate for non-grocery, non-software items.

Here is an outline of what you class should look like:

- Static Constants
  - double ITEM_TAX_RATE
- Instance Variables(make sure these are the correct visibility!)
  - String name
  - double price
  - double taxRate // 0.09 for example for 9% a tax rate
- Methods
  - Constructor
    - `public Item(String name, double price, double taxRate){…}`
    - `public Item(String name, double price){…}`

```
//this constructor should be called when you are making a
//new Item. Make sure you set the taxRate to the correct
//amount.  The child classes should call the other
//constructor as their super constructor in order to
//correctly set the taxRate
```

- method to get the Item's price
  - `public double getPrice(){…}`
- method to get the Item's price after tax
  - `public double getPriceAfterTax(){…}`
- method to get a String representation of the Item
  - `public String toString(){…}`

7.2 GroceryItem.java

GroceryItem should be a child class of Item. The properties of inheritance let us keep all of the functionality of the Item class when we extend it, so we only have to write a few additional things in GroceryItem.

Grocery Item should have a way to keep track of whether or not an item is perishable, and it should have its own version of the toString method that will include everything that is in Item's toString, as well as information about the perishability of the GroceryItem.

GroceryItem should also have a constant for the tax rate on GroceryItems – all GroceryItems should have the same tax. Use a value of 0.025 for the tax rate for grocery items. Note the standard tax rate does not apply, meaning a grocery item's total cost will be the grocery item's cost with the grocery tax added on. No item tax will be added on.

Here is an outline of the variables and methods your class should have:

- Static Constants
  - double GROCERY_TAX_RATE
- Instance Variables(make sure these are the correct visibility!)
  - boolean perishable
- Methods
  - Constructor
    - `public GroceryItem(String name, double price, boolean perishable){…}`

      ```
      // this constructor must use constructor chaining to the
      // superclass's constructor and also pass the
      // GROCERY_TAX_RATE using the constant you set up.
      ```
  - method to get the String representation of the GroceryItem – this should build off of Item's toString by using method chaining.
    - `public String toString(){…}`

7.3 SoftwareItem.java

SoftwareItem should also be a child class of Item.

SoftwareItem should have a way to keep track of what OS it is compatible with (this can just be a String), and it should have its own version of the toString method that will include everything that is in Item's toString, as well as information about the OS compatibility of the SoftwareItem.

SoftwareItem should also have a constant for the tax rate on SoftwareItems – all SoftwareItems should have the same tax rate of 0.15. Note the standard tax rate does not apply.

Here is an outline of the variables and methods your class should have:

- Static Constants
  - double SOFTWARE_TAX_RATE
- Instance Variables(make sure these are the correct visibility!)
  - String compatibility
- Methods
  - Constructor
    - `public SoftwareItem(String name, double price, String compatibility){…}`
      `// this constructor must use constructor chaining to the`
      `// superclass's constructor and also pass the`
      `// SOFTWARE_TAX_RATE using the constant you set up.`
  - method to get the String representation of the SoftwareItem – this should build off of Item's toString by using method chaining
    - `public String toString(){…}`

7.4 ShoppingCart.java

This class will use an array internally to represent a real world shopping cart. The purpose of ShoppingCart is to hold some number of Items. You are required to begin with a small array – literally length 5. If someone tries to add a 6th item, you are to create a new array with twice the length, copy all items of the old shopping cart array into the new one, and then finally successfully add that 6th item (and so on every time the internal array gets filled).

You are expressly required to use literally an array for this. Do not use an ArrayList, Vector, or any other data structure. And yes, literally start it with size 5. The point is for you to hone your array ninja skills!

The ShoppingCart will need to have a way to add items to the cart, remove the last item added to the cart (must be able to remove more than once), get all of the items out of the cart, and create a nicely formatted String describing the contents of the cart.

Here is an outline of the variables and methods your class should have:

- Instance Variables(make sure these are the correct visibility!)
    - Item[] items
    - int count // current number of items in the array
- Methods
    - Constructor
        - `public ShoppingCart(){…}`
    - method to add an item to the ShoppingCart
        - `public void addItem(Item item){…}`
    - method to remove the last Item added to the ShoppingCart
        - `public Item removeLastItem(){…}`
    - method to get the Items in the ShoppingCart
        - `public Item[] getItems(){…}`
    - method to get a String representation of the ShoppingCart
        - `public String toString(){…}`

7.5 Customer.java

This class represents the person who is doing the shopping. A Customer should have a name, a ShoppingCart to hold the items the Customer wishes to purchase, a way to keep track of how much money the Customer has, and a way to keep track of whether or not they are still shopping.

A Customer can add an item to their ShoppingCart, remove the last Item they added to their ShoppingCart, give the contents of their ShoppingCart to the cashier/program at checkout, and pay the cost indicated by the cashier/program.

The Customer should also have a way to set whether or not they are still shopping, and a way to get the money they have left. Finally, implement toString() to get a String representation of the Customer that states the amount of money the customer has and the items that are in their cart.

Here is an outline of the variables and methods your class should have:

- Instance Variables(make sure these are the correct visibility!)
    - String name
    - ShoppingCart cart
    - double money
    - boolean shopping
- Methods
    - Constructor
        - `public Customer(String name, double money){…}`
    - method to add an item to the Customer's cart

        - `public void addToCart(Item item){…}`
    - method to remove the last Item added to the Customer's cart
        - `public Item removeLastItem(){…}`
          `// note this must be able to work repeatedly meaning`
          `// after adding three items, you should be able`
          `// to remove, remove, and then still have that first`
          `// original item in the cart.`
          `// Also a remove on an empty cart should just do nothing.`
    - method to get the contents of the Customer's ShoppingCart for checkout
        - `public Item[] checkout(){…}`
    - method to remove the correct amount of money when asked to pay
        - `public boolean pay(double cost){…}`
        - this should return true if the Customer had enough money to cover the cost, and false if they did not
    - method to set whether or not the Customer wishes to continue shopping
        - `public void keepShopping(boolean shopping){…}`
    - method to get whether or not the Customer is shopping
        - `public boolean isShopping(){…}`
    - method to get the amount of money the Customer has
        - `public double getMoney(){…}`
    - method to get a String representation of the Customer (includes money and shopping cart items)
        - `public String toString(){…}`

7.6 Store.java

The Store class represents the actual store where a person can shop. A store has a name and an inventory of items – internally stored as an array. This array should also initially be of length 5. And again, you will need to handle resizing the array when more Items are added to it than will fit in the current array. It keeps track of the number of items that it has, and the amount that it has earned from selling items.

A Store is able to receive shipments of Items to be added into the Store's inventory. A Store is able to get an Item for a Customer from the inventory, and it should be able to put back an Item that the customer does not want. It can ring up a Customer's items and charge the Customer for the correct amount including the appropriate taxes. If the Customer cannot afford the items they wish to buy, the Store should be able to add the contents of that Customer's cart back into inventory.

Store will have a toString() method to provide a String representation of the Store, including its name, it's inventory (the items that is has), and of the store itself (how many total items it has left, and how much it has earned). The Store also can determine if it has any items left.

Here is an outline of the variables and methods your class should have:

- Instance Variables(make sure these are the correct visibility!)
    - Item[] inventory
    - int count // number of items in inventory
    - double moneyEarned
    - String name
- Methods
    - Constructor
        - `public Store(name){…}`
    - method to add Items to the Store's inventory
        - ```
          public void receiveShipment(Item[] items){…}
          // Notice this must work even if more items come in
          // than will fit in the array.  You are to create a new
          // array with double the length, copy all old inventory
          // over, and then copy in the new inventory when necessary.
          ```
    - method to get an Item out of inventory
        - `public Item getItem(int index){…}`
        - this is how the Customer will interact with the Store's inventory – through the driver, we will prompt the Customer to enter an index that corresponds to a particular item
    - method to ring up the Customer's items and handle payment
        - `public boolean makeASale(Customer Customer){…}`
        - Note: This could be one method or broken into separate ones – it is up to you. Do not make any static methods though other than main. Use proper OOP design.
    - method to determine if Store has any items left to sell
        - `public boolean hasMoreItems(){…}`
    - method to get a String representation of the Store's inventory
        - `public String listInventory(){…}`
    - method to get a String representation of the Store
        - `public String toString(){…}`
          ```

7.7 StoreDriver.java

This class will be where we actually create all of the elements of our stores and our customers, and allow them to interact.

The main things that should happen in this class are as follows:

- create a Store
- create a Customer
- create a second Store
- create a second Customer
- create at least 6 different Items for Store 1
    - you should have at least 1 of every type (Item, SoftwareItem, and GroceryItem)
- add the Items into the Store 1's inventory
- create at least 6 different Items for Store 2
    - again have at least 1 of every type (Item, SoftwareItem, and GroceryItem)
- add the Items into the Store 2's inventory
- prompt one of the Customers to buy things from one of the stores until they quit shopping, or the store runs out of items to sell
    - you should always tell the user
        - how many Items the Store has available
        - how much it has made off of selling its Items
        - what Items the Store has available
        - what number to type to get that Item
        - the shopper's current cart and money information
    - if a Customer cannot afford the contents of their ShoppingCart, the contents should be put back into the Store's inventory
    - if the Customer has placed all of the Store's inventory into their ShoppingCart, they should be prompted to buy everything.
- basically go through a shopping experience by prompting the other Customer using the other Store in a likewise fashion.
- before ending the program print one last time the Store info for both stores.

The resulting output from this class should look something like this:

```
Welcome!
-----------------------------------------------------
The Tenacious TA's Tradehouse has 6 items. It has made $0.00 from sales.

Currently, we have
0: Item Name: Coffee  Price: $1.00  Perishable: yes
1: Item Name: Super Mario Bros  Price: $10.00  Compatable OS: Super Nintendo
2: Item Name: Software Design Practices  Price: $20.00
3: Item Name: Scone  Price: $2.50  Perishable: yes
4: Item Name: Gummy Snacks  Price: $1.00  Perishable: no
5: Item Name: Dry Erase Markers  Price: $6.00

If you would like to buy something, enter the number that corresponds with the item
to add it to your cart.
If you do not see anything you would like, enter -1
0
Bill, you currently have $36.00.
The items in your cart are:
Item Name: Coffee  Price: $1.00  Perishable: yes

If you like to checkout, please type 'yes'
If you would like to abandon your shopping cart, please type 'leave'
If you would like to remove the last item from your cart, please type 'remove'
If you would like to add more items, please type another word or letter
no


-----------------------------------------------------
The Tenacious TA's Tradehouse has 5 items. It has made $0.00 from sales.

Currently, we have
0: Item Name: Super Mario Bros  Price: $10.00  Compatable OS: Super Nintendo
1: Item Name: Software Design Practices  Price: $20.00
2: Item Name: Scone  Price: $2.50  Perishable: yes
3: Item Name: Gummy Snacks  Price: $1.00  Perishable: no
4: Item Name: Dry Erase Markers  Price: $6.00

If you would like to buy something, enter the number that corresponds with the item
to add it to your cart.
If you do not see anything you would like, enter -1
0
Bill, you currently have $36.00.
The items in your cart are:
Item Name: Coffee  Price: $1.00  Perishable: yes
Item Name: Super Mario Bros  Price: $10.00  Compatable OS: Super Nintendo

If you like to checkout, please type 'yes'
If you would like to abandon your shopping cart, please type 'leave'
If you would like to remove the last item from your cart, please type 'remove'
If you would like to add more items, please type another word or letter
no
```

```
-----------------------------------------------------
The Tenacious TA's Tradehouse has 4 items. It has made $0.00 from sales.

Currently, we have
0: Item Name: Software Design Practices  Price: $20.00
1: Item Name: Scone  Price: $2.50  Perishable: yes
2: Item Name: Gummy Snacks  Price: $1.00  Perishable: no
3: Item Name: Dry Erase Markers  Price: $6.00

If you would like to buy something, enter the number that corresponds with the item
to add it to your cart.
If you do not see anything you would like, enter -1
1
Bill, you currently have $36.00.
The items in your cart are:
Item Name: Coffee  Price: $1.00  Perishable: yes
Item Name: Super Mario Bros  Price: $10.00  Compatable OS: Super Nintendo
Item Name: Scone  Price: $2.50  Perishable: yes

If you like to checkout, please type 'yes'
If you would like to abandon your shopping cart, please type 'leave'
If you would like to remove the last item from your cart, please type 'remove'
If you would like to add more items, please type another word or letter
yes
Thank you for your purchase! You have $20.93 left.Would you like to keep shopping?
no


Goodbye!
Welcome!
-----------------------------------------------------
The Super Student Store has 6 items. It has made $0.00 from sales.

Currently, we have
0: Item Name: Orange  Price: $1.50  Perishable: yes
1: Item Name: Bejewled  Price: $10.00  Compatable OS: Windows
2: Item Name: Mystery Novel  Price: $12.00
3: Item Name: Pasta Sauce  Price: $2.50  Perishable: no
4: Item Name: Yogurt  Price: $1.00  Perishable: yes
5: Item Name: Coffee Cup  Price: $6.00

If you would like to buy something, enter the number that corresponds with the item
to add it to your cart.
If you do not see anything you would like, enter -1
0
Alice, you currently have $30.00.
The items in your cart are:
Item Name: Orange  Price: $1.50  Perishable: yes

If you like to checkout, please type 'yes'
If you would like to abandon your shopping cart, please type 'leave'
If you would like to remove the last item from your cart, please type 'remove'
If you would like to add more items, please type another word or letter
no
```

```
-----------------------------------------------------
The Super Student Store has 5 items. It has made $0.00 from sales.

Currently, we have
0: Item Name: Bejewled  Price: $10.00  Compatable OS: Windows
1: Item Name: Mystery Novel  Price: $12.00
2: Item Name: Pasta Sauce  Price: $2.50  Perishable: no
3: Item Name: Yogurt  Price: $1.00  Perishable: yes
4: Item Name: Coffee Cup  Price: $6.00

If you would like to buy something, enter the number that corresponds with the item
to add it to your cart.
If you do not see anything you would like, enter -1
3
Alice, you currently have $30.00.
The items in your cart are:
Item Name: Orange  Price: $1.50  Perishable: yes
Item Name: Yogurt  Price: $1.00  Perishable: yes

If you like to checkout, please type 'yes'
If you would like to abandon your shopping cart, please type 'leave'
If you would like to remove the last item from your cart, please type 'remove'
If you would like to add more items, please type another word or letter
remove


-----------------------------------------------------
The Super Student Store has 5 items. It has made $0.00 from sales.

Currently, we have
0: Item Name: Bejewled  Price: $10.00  Compatable OS: Windows
1: Item Name: Mystery Novel  Price: $12.00
2: Item Name: Pasta Sauce  Price: $2.50  Perishable: no
3: Item Name: Coffee Cup  Price: $6.00
4: Item Name: Yogurt  Price: $1.00  Perishable: yes

If you would like to buy something, enter the number that corresponds with the item
to add it to your cart.
If you do not see anything you would like, enter -1
3
Alice, you currently have $30.00.
The items in your cart are:
Item Name: Orange  Price: $1.50  Perishable: yes
Item Name: Coffee Cup  Price: $6.00

If you like to checkout, please type 'yes'
If you would like to abandon your shopping cart, please type 'leave'
If you would like to remove the last item from your cart, please type 'remove'
If you would like to add more items, please type another word or letter
yes
Thank you for your purchase! You have $21.93 left.Would you like to keep shopping?
yes
```

```
-----------------------------------------------------
The Super Student Store has 4 items. It has made $8.07 from sales.

Currently, we have
0: Item Name: Bejewled  Price: $10.00  Compatable OS: Windows
1: Item Name: Mystery Novel  Price: $12.00
2: Item Name: Pasta Sauce  Price: $2.50  Perishable: no
3: Item Name: Yogurt  Price: $1.00  Perishable: yes

If you would like to buy something, enter the number that corresponds with the item
to add it to your cart.
If you do not see anything you would like, enter -1
0
Alice, you currently have $21.93.
The items in your cart are:
Item Name: Bejewled  Price: $10.00  Compatable OS: Windows

If you like to checkout, please type 'yes'
If you would like to abandon your shopping cart, please type 'leave'
If you would like to remove the last item from your cart, please type 'remove'
If you would like to add more items, please type another word or letter
yes
Thank you for your purchase! You have $10.43 left.Would you like to keep shopping?
yes


-----------------------------------------------------
The Super Student Store has 3 items. It has made $19.57 from sales.

Currently, we have
0: Item Name: Mystery Novel  Price: $12.00
1: Item Name: Pasta Sauce  Price: $2.50  Perishable: no
2: Item Name: Yogurt  Price: $1.00  Perishable: yes

If you would like to buy something, enter the number that corresponds with the item
to add it to your cart.
If you do not see anything you would like, enter -1
0
Alice, you currently have $10.43.
The items in your cart are:
Item Name: Mystery Novel  Price: $12.00

If you like to checkout, please type 'yes'
If you would like to abandon your shopping cart, please type 'leave'
If you would like to remove the last item from your cart, please type 'remove'
If you would like to add more items, please type another word or letter
leave


Goodbye!
```

7.8 Testing

A huge part of software development is testing. You always want to make sure that your code works (no crashing), and that it works in the way you intend it to.

For this program, make sure you test all possible cases you can think of - don't forget the edge cases! Even things like actually the customer decided to buy nothing.

In a few days, we will be posting a few files that contain the results of our own tests. At the top of the file we will describe what items we put into the store, what our tax rates were, and how much money the Customer had. You will be able to run your code in the same way, and see if you get the same results that we did. You should also test out things as much as you can on your own. Our tests will not be exhaustive. The TAs will use these tests and more then they test your code.

Here is a list of important things to test for and edge cases to consider:

- adding items to the store's inventory
- customer picking an item out of the store
- customer removing the last item from their cart
- customer checkout
    - with enough money to cover their items
    - without enough money to cover their items
- customer buying everything in the store
    - with enough money to cover it
    - without enough money to cover it
- customer putting everything in the store in their cart and then removing an item
- customer buying nothing
- customer picking one item, then removing that last item and then checking out
- weird cases your code must handle
    - customer tries to pick a number that does not correspond to an item on the list – just remind them that number is not valid and reshow the menu of items letting them chose again.
    - customer tries to remove their last item yet they have no items left. Just let this slide. The program should not fail, but no change to the cart occurs.
    - customer adds 3 items and then removes twice. The cart should still have that original item in it.
    - customer has no items in their cart and they check out
    - customer added one item, removed it, and then checked out
    - customer adds more items than what will fit originally in the internal array (more than 5 items) – the internal array must be replaced with a bigger one, etc.
    - store adds inventory that will not fit in its internal array. A new array twice the size must be created, filled, etc.

Turn-in Procedure

Turn in the following files to T-Square. When you are ready, make sure that you have actually **submitted** your files, and not just saved them as a draft.

- Item.java
- GroceryItem.java
- SoftwareItem.java
- ShoppingCart.java
- Customer.java
- Store.java
- StoreDriver.java

Note** Always submit .java files - never submit your .class files. Once you have submitted and received the email from T-Square, you should download your files into a fresh folder. Compile, run, and test those exact files that you turned in. It is pretty much foolproof if you use this technique. Anything less than this is a gamble. See "safe submission" info below. File issues, non-compiling files, non-source code files, etc are all problems and will cause a 0 for the HW. Also, make sure that your files are in on time; the real deadline is 8 pm. While you have until 2 am to get it turned in, we will not accept homework past 2 am for any reason. Don't wait until the last minute!

Verify the Success of Your HW Turn-in

Practice "safe submission"! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.
2. Read that email. Look at those filenames. We do not grade .class files. We require source code .java files.
3. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files and the fact that you submitted.
4. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.
5. Recompile and test those exact files.
6. This helps guard against a few things.
    a. It helps insure that you turn in the correct files.
    b. It helps you realize if you omit a file or files. (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)
    c. Reading the email and looking at the files you download helps prevent the turn-in of bytecode (.class) file for which you will receive no credit.
    d. Helps find last minute causes of files not compiling and/or running.