COMPUTER SCIENCES AND SOFTWARE ENGINEERING
AUBURN UNVERSITY

COMP 2710
Software Construction

Summer 2012

Lab 4
Simple Cash Register Application

Due: November 28, 2012

Points Possible: 100
Due: Written Portion: Nov. 14th, 2012 by 11:55 pm. Submission via Canvas (25 points)
Program: November 28th, 2012 by 11:55 pm via Canvas (75 points)

No late assignments will be accepted.
No collaboration between students. Students should NOT share any project code with each other. Collaborations in any form will be treated as a serious violation of the University's academic integrity code.

Goals:

- To develop a simple cash register application taking advantage of inheritance
- To use virtual functions to simplify implementation
- To understand the use of vectors that contain different type of objects
- To perform Object-Oriented Analysis, Design, and Testing
- To learn the use of vector class template

Process – 25 points:
Create a text, doc, or .pdf file named "<username>-4p" (for example, mine might read "lim-4p.txt") and provide each of the following. Please submit a text file, a .doc file or .pdf file (if you want to use diagrams or pictures, use .doc or .pdf). You are free to use tools like Visio to help you, but the final output needs to be .txt, .doc, or .pdf.

1. **Analysis:** Prepare use cases. Remember, these use cases describe how the user interacts with a cash register system (what they do, what the systems does in response, etc.). Your use cases should have enough basic details such that someone unfamiliar with the system can have an understanding of what is happening in the cash register system interface. They should not include internal technical details that the user is not (and should not) be aware of. *You must include use a case diagram show relationships between use cases. Your use case descriptions must include the following: Use case name, description, assumptions, actors, pre-conditions, post-conditions, steps (basic flow), variations (alternate flow), non-functional requirements, and issues.*

2. **Design**:

a. Create a Class Diagram. Be sure to include:

1) The name and purpose of the classes
2) The member variables and the functions of the class
3) Show the interactions between classes (for example, ownership or dependency)
4) Any relevant notes that don't fit into the previous categories can be added

b. Create the data flow diagrams. Show all the entities and processes that comprise the overall system and show how information flows from and to each process.

3. **Testing**: Develop lists of <u>*specific*</u> test cases OR a driver will substitute for this phase:

1) For the system at large. In other words, describe inputs for "nominal" usage. You may need several scenarios. In addition, suggest scenarios for abnormal usage and show what your program should do (for example, entering a negative number for a menu might ask the user to try again).
2) For each object. (Later, these tests can be automated easily using a simple driver function in the object)

4. Implement the simple cash register application

5. **Test Results**: ***After developing the cash register system***, actually try all of your test cases (both system and unit testing). Show the results of your testing (a copy and paste from your program output is fine – don't stress too much about formatting as long as the results are clear). You should have test results for every test case you described. If your system doesn't behave as intended, you should note this. Note: Driver output will substitute for this phase.

<u>Program Portion:</u>

In this Lab assignment, you will implement a simple cash register application that allows the cashier to perform the following functions:

1. Enter all items purchased and their prices
2. Compute different types of sales, including regular sales, discount sales, and mail-order sales, and then calculates the sub-total amount
3. Calculate tax from the sub-total amount to obtain the total amount payable
4. Process different types of payments, including cash, check and credit card payments
5. Print the receipts for different types of sales and payments in (2) and (3) above

You will use inheritance and virtual functions to simplify the definition of the classes that contain the above variations in functions for each of the different sales types: cash, check and credit card sales.

In Function (1), the cashier must itemize all the items bought by the customer with their prices and calculate the subtotal. When all the items have been entered, the cashier can end the itemization using the special option '*'.

In Function (2), the sales must be one of the three sales types: regular sale, discount sale and mail-order sale. In regular sale, the sub-total calculated in Function (1) above is used for the next function. In discount sales, a discount of 10% is deducted from the previous subtotal from Function (1) and a new sub-total is used. In mail-order sales, a shipping and handling charge of $3.50 is added to each item in the list of items. You can implement each of the three different types of sale as derived class from a base `sale_type` class.

In Function (3), a 7% tax will be calculated from the sub-total to obtain the total amount.

Function (4) is called `process_payment` and is used for processing different types of payments, including cash, check and credit card payments. It must be implemented as virtual function, where the function `process_payment` must be re-defined in the three derived classed to process cash, check and credit card payments. In cash payments, the cashier will enter the cash amount received from the customer and then computes the change. In check payment, the cashier must enter the name on the check and the driver's license number. In credit card payment, the cashier must enter the name on the card, expiration date, and credit card number. All these sales information for each customer will be stored in member variables of the sales object that is of a type based on the types of payment, i.e. cash, check or credit card sales.

In Function (5), the `print_sale` function will first print the sale banner and number and then prints the items, sales types (regular, discount or mail-order information), subtotal, tax, etc. Then it must call the virtual function `print_payment` to print the payment information. The `print_payment` function must be a virtual function that is actually defined in the overriding virtual function of the derived cash, check and credit card sales classes to print the appropriate payment information.

You must define the following classes: `Sale, CashSale, CheckSale and CreditCardSale.` These classes must contain member functions and variables as described below.

First, define a base class named `Sale` that contains member variables which store the *list of item description and price, sale types information (discount or mail-order), sales tax,* the *total amount, and payment information.* Also, define the appropriate accessor and mutator functions for `Sales`. Next, create a ***virtual*** member function named `process_payment` that process the appropriate payment, where the virtual function is re-defined in the derived classes for different sale types to process the payment type correctly. This class must also have the `print_sale` function to print the descriptions of the *list of item description and price*, sale types (e.g. discount), *sales tax, total amount and the payment information.* The `print_sale` function **must call the virtual `print_payment` function** to print the appropriate payment information based on whether the sale object is of the cash, check and credit card sales classes.

Next, define a class named `CashSale` that is derived from the `Sale` class. This class must define the overriding ***virtual function*** `processs_payment` to enter and store the amount of cash received and calculate and store the change. It must also define the overriding ***virtual function*** `print_payment` to print the amount of cash received and the change. Include appropriate constructor(s).

Define a class named `CheckSale` that is derived from the `Sale` class. This class must contain member variables for the name on the check, and driver's license number. Include appropriate constructor(s). This class must define the overriding ***virtual function*** `processs_payment` to enter and store the name on the check and the driver's license number. It must also define overriding the ***virtual function*** `print_payment` to print the check payment information.

Finally, define a class named `CreditCardSale` that is derived from the `Sale` class. This class must contain member variables for the name on the card, expiration date, and credit card number. Include appropriate constructor(s). This class must define the overriding ***virtual function*** `processs_payment` to enter and store the name on the card, expiration date, and credit card number. It must also define the overriding ***virtual function*** `print_payment` to print the credit card payment information.

Your program must keep *a single list* of different types of sales in a vector of pointers to `Sale` objects. The program will repetitively do one of two things: process payment or print sales from the sales list. It will continuously prompt the user for payment(p) or print sales (s).

If the option is to process payment, the program will continuous prompt to itemize all the items bought by the customer with their prices and calculate the subtotal. When all the items have been entered, the cashier can end the itemization using the special option '*'. It will then perform Function (2) to process on of the three types of sales: regular sale, discount sale and mail-order sale. It then performs Function (3) to add a 7% tax and calculate the total amount payable. It then calls `process_payment` tor process different types of payments, including cash, check and credit card payments.

If the option is to print sales, your program must be able to print all the sales in the vector. Although the vector contains pointers to the base class `Sale`, they actually points to the derived class `CashSale, CheckSale, CreditCardSale` objects. So when the function `print_sale` is called, it in turn calls `print_payment.` Since `print_payment` is virtual, the actual `print_payment` function of the derived class is called.

The user interface

Write a menu-based and text-based user interface for the simple cash register application where the cashier can enter either cash payment or credit card payments. Each payment will be stored in the system in a vector called `saleslist`. At any point the cashier can print out all the sales stored in the cash register.

The following is a sample of a program that uses the API provided by the `Payment` class.

Sample Usage of the API:

```
int main()
{
/*
Create a vector or "basket of sales"
Note: The data type is a pointer to the BASE CLASS Sale
*/
vector<Sale *> saleslist;

//create some objects of the various derived classes

CashSale* cs = new CashSale();
/*
The codes that follow will set the appropriate cash payment
amount, change, etc. entered by the cashier.
Then, add the derived class objects to the vector.
This is legal, since saleslist stores pointers to Sale
objects.
*/
saleslist.push_back(cs);

CheckSale* cks = new CheckSale();
/*
The codes that follow will set the appropriate check
payment amount, name, driver's license number, etc. entered
by the cashier.
Then, add the pointer to the derived class objects to the
vector.
*/
saleslist.push_back(cks);

CreditCardSale* ccs = new CreditCardSale();
/*
The codes that follow will set the appropriate credit card
payment amount, etc. entered by the cashier.
Then, add the pointer to the derived class objects to the
vector.
*/
saleslist.push_back(ccs);

/*
Call the function print_sale in the base class Sale which
will call the virtual function print_payment() as defined
in the CashSale class to output the complete payment
information on the cash payment.
*/
saleslist[0]->print_sale();

/*
Call the function print_sale in the base class Sale which
will call the virtual function print_payment() as defined
in the CheckSale class to output the complete payment
information on the check payment.
*/
saleslist[1]->print_sale();

/*
Call the function print_sale in the base class Sale which
will call the virtual function print_payment() as defined
in the CreditCardSale class to output the complete payment
information on the credit card payment.
*/
saleslist[2]-> print_sale();

}
```

The above sample code does not show the menu and how the appropriate payment values and information are entered by the user into the payment objects. In your program, you must integrate all these operations.

The user interface must check for correct input value from the users. If there is any error, e.g. selecting an invalid payment value, then the program must display the appropriate error message and continue to prompt for the correct input. Your program must not exit or terminate when there is an incorrect input value.

The name of your program must be called <username>_4.cpp (for example, mine would read "lim_4.cpp"

Use comments to provide a heading at the top of your code containing your name, Auburn Userid, and filename. You will lose points if you do not use the specific program file name, or do not have a comment block on **EVERY** program you hand in.

Your program's output need not exactly match the style of the sample output (see the end of this file for one example of sample output).

Important Notes:
You must use an object-oriented programming strategy in order to design and implement this cash register system (in other words, you will need write class definitions and use those classes, you can't just throw everything in main()). A well-done implementation will produce a number of robust classes, many of which may be useful for future programs in this course and beyond. Some of the classes in your previous lab project may also be re-used here, possibly with some modifications. Remember good design practices discussed in class:

a) A class should do one thing, and do it well
b) Classes should NOT be highly coupled
c) Classes should be highly cohesive

- You should follow standard commenting guidelines.

- You DO NOT need any graphical user interface for this simple, text-based application. If you want to implement a visualization of some sort, then that is extra credit.

Error-Checking:

You should provide enough error-checking that a moderately informed user will not crash your program. This should be discovered through your unit-testing. Your prompts should still inform the user of what is expected of them, even if you have error-checking in place.

Submit your program through the Canvas online system. If for some disastrous reason Canvas goes down, instead e-mail your submission to TA – Dongjin Kim – at dzk0011@tigermail.auburn.edu. Canvas going down is not an excuse for turning in your work late.

You should submit the two files in digital format. No hardcopy is required for the final submission:

<username>_4.cpp
<username>_4p.txt (script of sample normal execution and a script of the results of testing)

A sample execution is shown below, where the bold fonts indicate input by the user.

```
>  lim_4

        ============================================================
        |                 Welcome to Great Buy!                    |
        ============================================================

Enter payment(p) or print sales(s):  p

Enter item:  Calculator

Enter amount:  $ 35.70

Enter item:  Battery

Enter amount:  $ 5.20

Enter item:  *

Sub-Total:  $ 40.90

Enter type of Sale: regular(r), discount(d) or mail-order(m): r

Tax:  $ 2.86

Total amount:  $ 43.76

Enter type of payment, cash(c), check(k) or credit card (d):  c

Amount received:  $ 50.00

Change:  $ 6.24


        ============================================================
        |                 Welcome to Great Buy!                    |
        ============================================================

Enter payment(p) or print sales(s):  p
Enter item:  Printer

Enter amount:  $ 148.30

Enter item:  Ink Catridge

Enter amount:  $ 43.70

Enter item:  *

Sub-Total:  $ 192.00

Enter type of Sale: regular(r), discount(d) or mail-order(m): m
Shipping and handling: $ 7.00

Sub-Total:  $ 199.00

Tax:  $ 13.93

Total amount:  $ 212.93

Enter type of payment, cash(c), check(k) or credit card (d):  d

Enter name on the credit card:  John Doe

Enter credit card number:  123456782468

Enter expiration date:  07/12


        ==========================================================
        |                 Welcome to Great Buy!                  |
        ==========================================================

Enter payment(p) or print sales(s):  p

Enter item:  Scanner

Enter amount:  $ 105.70

Enter item:  Printer paper

Enter amount:  $ 21.40

Enter item:  *

Sub-Total:  $ 127.10

Enter type of Sale: regular(r), discount(d) or mail-order(m): d

Discount: $12.70

Sub-Total:  $ 114.40

Tax:  $ 8.01
```

```
Total amount:  $ 122.41

Enter type of payment, cash(c), check(k) or credit card (d):  k

Enter name on the check:  Jane Smith

Enter driver's license number:  24680135


        ===========================================================
        |                 Welcome to Great Buy!                   |
        ===========================================================

Enter payment(p) or print sales(s):  s

Sales #1:
   1. Calculator  $ 35.70
   2. Battery  $ 5.20
      Sub-Total:  $ 40.90
      Tax:  $ 2.86
      Total amount:  $ 43.76
      CASH
      Amount received:  $ 50.00
      Change:  $ 6.24

Sales #2:
   1. Printer  $ 148.30
   2. Ink Catridge  $ 43.70
      Sub-Total:  $ 192.00
      Shipping and handling: $ 7.00
      Sub-Total:  $ 199.00
      Tax:  $ 13.93
      Total amount:  $ 212.93
      CREDIT CARD John Doe 123456782468 07/12

Sales #3:
   1. Scanner     $ 105.70
   2. Printer paper   $ 21.40
      Sub-Total:  $ 127.10
      Discount: $12.70
      Sub-Total:  $ 114.40
      Tax:  $ 8.01
      Total amount:  $ 122.41
      CHECK   Jane Smith 24680135

End

        ===========================================================
        |                 Welcome to Great Buy!                   |
        ===========================================================

Enter payment(p) or print sales(s):
```