Programming Assignment #4

Learning objective: Upon completion of this program, you should be able to work with dynamic arrays of objects, as well as the use of objects from more than one class, using the composition ("has-a") relationship. This program should be portable, so it will work with any C++ compiler. Make sure that it works with the g++ compiler and with Visual C++ before you hand it in.

Description:

You will be writing classes that simulate sales at a store that sells books, DVDs, and software, as well as the accounting of the day's sales on one cash register. The register will keep track of a list of the most recent sales. You will need to finish the writing of two classes: Sale and Register. The full header file for the Sale class has been provided in a file called sale.h. The file is located as an appendix at the end of this assignment.

Program Details and Requirements:

Sale class

Using the Sale class declaration, write the **sale.cpp** file and define all of the member functions that are declared in the file sale.h. (Do not change sale.h in any way. Just take the existing class interface and fill in the function definitions). Notice that there are only four kinds of items: BOOK, DVD, SOFTWARE, and CREDIT. The sale.h file contains the descriptions of how the functions should work and what the member data variables represent. Make sure that the member functions only do their intended tasks (i.e. no extraneous inputs, outputs, or results that are not specified).

Register class

Write a class called Register (filenames are **register.h** and **register.cpp**). A Register object should store at least the following information: an identification number (an integer) for the cash register object, the amount of money in the cash register object, and a list of sales. All member data must be private.

There is **no size limit** to the sales list, so it should be implemented with a dynamically allocated array. (This means you'll need an array of Sale objects). There should never be more than 5 unused slots in this array (i.e. the number of allocated spaces may be **at most** 5 larger than the number of slots that are actually filled with real data). This means that you will need to ensure that the array allocation expands or shrinks at appropriate times. Whenever the array is resized, print a message (for testing purposes) that states that the array is being resized, and what the new size is. Example: "** Array being resized to 10 allocated slots". Correct memory management should be used, including the cleanup of memory (i.e. you must program the class so that it leaves no "memory leaks"). This means that you need to deallocate any dynamic space, when and where appropriate.

You can add any private data and functions into the Register class that you feel are useful, but your Register class must have at least the following functions (names and descriptions provided here) in its public interface. These functions should not do extraneous input, output, or other non-specified tasks (i.e. they do only what is specified).

(Hint: keep in mind that inside the register, you are keeping a dynamic array of Sale objects. This means that most of these functions will be using this array to do their work -- and they can also call upon Sale class member functions).

Constructor

The constructor should take in two parameters, which allow the ID number and starting amount in the register to be initialized when the Register object is created. To create a Register object, this data *must* be passed in.

the destructor

The destructor should do any appropriate clean-up tasks needed to manage the dynamic memory

- GetID
- GetAmount

These accessor functions should return the current ID number and current amount in the register, respectively, to the caller.

- RingUpSale

This function allows the item type and base price of a sale to be passed in as parameters. This function should store the sale in the sale list, and it should update the amount of money in the cash register appropriately. Items that are bought will add money to the register. Remember that sales tax must be added to the base price of any item sold. If the sale type is CREDIT, then you should deduct the amount from the register.

- ShowLast // display the last sale made
- ShowAll // display entire sale list

These functions will display information about sales to the screen. ShowLast should display only the information about the last sale that was made. ShowAll should show **all** of the sales currently in the sale list, one per line. This display should be in the order oldest to newest. (i.e. the most recent sale is displayed last). If there are no sales currently in the list, then output an appropriate message instead (like "No sales have been made").

- Cancel

This function should cancel the last sale in the list. This means that the amount in the cash register, as well as the data in the sale list, should be adjusted accordingly (as if the sale had never happened). This simulates the idea that a sale rung up incorrectly can be voided out by the cashier. If the sale list is empty, print an appropriate error message and abort the cancel operation.

- SalesTax

This function should calculate and return the total amount of sales tax for the **last** n sales (i.e. the most recent ones), where n is an integer passed in as a parameter. If n is higher than the number of sales currently stored, just calculate the total tax for **all** sales currently in the sale list. If n is invalid (a negative number), print an appropriate error message and return 0.

Grading Criteria:

- The program compiles. If the program does not compile no further grading can be accomplished. Programs that do not compile will receive a zero.
- (25 Points) The program executes without exception and produces output. The grading of the output cannot be accomplished unless the program executes.
- (25 Points) The program produces the correct output.
  - (5 points) ShowLast() works properly and displays correct results
  - (5 Points) ShowAll() works properly and displays correct results
  - (5 Points) Shows the current amount in the cash register
  - (5 Points) Displays the Total Sales tax for recent sales.
  - (5 Points) Message is displayed when it is resizing.
- (25 Points) The program specifications are followed.
  - (5 Points) Only four items can be stored (BOOK, DVD, SOFTWARE, AND CREDIT)
  - (5 Points) All constructors are defined and work properly
  - (5 Points) Dynamic storage is allocated properly
  - (5 Points) Destructor is defined properly and works.
  - (5 Points) All functions are defined properly and work as defined.
- (10 Points)The program is documented (commented) properly and no global variables are used.
- (5 Points)Use constants when values are not to be changed
- (5 Points)Use proper indentation
- (5 Points)Use good naming standards

Test Program

A main routine with a menu program that will help you interactively test the features of your Register class has been provided. It is located in Appendix B of this assignment.

This menu program will start by prompting the user to input a cash register ID and the starting amount in the register. Then it will create a Register object and enter a menu loop -- the menu options will work with both lower and upper case inputs:

```
  S:   Show current amount in the cash register
  R:   Ring up a sale
  D:   Display the last sale
  L:   Display the entire sale List
  C:   Cancel the last stored sale
  T:   Find the Total sales tax for recent sales
  M:   Show this Menu
  X:   eXit the program
```

Some of the menu options will prompt for other inputs. Each will allow you to interactively test the behavior of your Register class public interface. The code in `menu4.cpp` also demonstrates appropriate calls to the Register class member functions.

Submit the following files (using the usual web submission procedure):

```
  sale.cpp
  register.h
  register.cpp
```

Appendix A sale.h File:

```
// *********************************************************************
// Class: Sale
// File: sale.h -- header file for the Sale class
// Description: An object of type Sale will store information about a single
// sale.   The variable "item" stores the type of item being sold (one of the
// four  items in the enumerated type ItemType).  Books, music, and software
// are the types of items sold.  CREDIT stands for store credit, which incurs
// a negative monetary charge.
// The variable "price" stores the base price of the item
// The variable "tax" is used to store the 7% sales tax
// The variable "total" is used to store the final charge (price + tax)
// Date: December, 2007
// Author: Mr. Bob Myers
// *************************************************************************

enum ItemType {BOOK, DVD, SOFTWARE, CREDIT};

class Sale
{
public:
   Sale();                      // default constructor,
                                // sets numerical member data to 0

   void MakeSale(ItemType x, double amt);

        // the MakeSale function loads data into the Sale object.  The
        // item and price are passed in.  Compute tax and total.
        // (For store credits, tax is 0).

   ItemType Item();             // Returns the type of item in the sale
   double Price();              // Returns the price of the sale
   double Tax();                // Returns the amount of tax on the sale
   double Total();              // Returns the total price of the sale
   void Display();              // outputs sale info (described below)

  // for the Display function, output the sale item, the price, the tax,
  // and the total (all on one line).  For store credit, the amounts
  // should be enclosed in < > symbols, to indicate a negative charge.
  // All monetary output should be in dollars and cents format, to two
  // decimal places.
  // Examples: Book         $ 20.00    Tax:  $ 1.40   Total:  $ 21.40
  //           Credit      <$ 15.00>   Tax:  $ 0.00   Total: <$ 15.00>

private:
   double price;       // price of item or amount of credit
   double tax;         // amount of sales tax (does not apply to credit)
   double total;       // final price once tax is added in.
   ItemType item;      // transaction type
};
```

Appendix B: Sample Main Program

```
include <iostream>
#include <iomanip>
#include <cctype>
#include "register.h"

using namespace std;

void ShowMenu()
{
        cout << "\nPlease select one of the following options\n";
        cout << "    by pressing the indicated key:\n";
        cout << "\tS: Show current cash register amount\n";
        cout << "\tR: Ring up a sale\n";
        cout << "\tD: Display the last sale\n";
        cout << "\tL: Display all sales in List\n";
        cout << "\tC: Cancel the last sale\n";
        cout << "\tT: Find total sales Tax for recent sales\n";
        cout << "\tM: Show this Menu\n";
        cout << "\tX: eXit the program\n";
}

int main()
{
   int num, id;
   double cash, amt;
   char kind, choice = 'X';

   cout << "\nEnter cash register id: ";
   cin >> id;
   cout << "\nEnter starting amount in register: ";
   cin >> cash;

   Register a(id,cash);
   ShowMenu();

   cout << setprecision(2) << fixed;  // set print flags

   do
   {
        cout << "\n> ";                 // Prompt the user,
        cin >> choice;                  // get a new command,
        choice = toupper(choice);       // and convert it to upper-case
        cin.get();                      // extract white space

        switch (choice)
        {
           case 'S':
                cout << "\nRegister # " << a.GetID() << "  Balance = $ "
                  << a.GetAmount() << endl;
                break;

           case 'R':
                cout << "\nWhat kind of sale?\n";
                cout << "(b)ook, (d)vd, (s)oftware, or (c)redit: ";
                cin >> kind;
                cout << "\nPlease enter the price: ";
                cin >> amt;
                if (kind == 'b')        a.RingUpSale(BOOK,amt);
                else if (kind == 'd')   a.RingUpSale(DVD,amt);
                else if (kind == 's')   a.RingUpSale(SOFTWARE,amt);
                else if (kind == 'c')   a.RingUpSale(CREDIT,amt);

                break;

           case 'D':
                a.ShowLast();           break;
           case 'L':
                a.ShowAll();            break;
           case 'C':
                a.Cancel();
                cout << "\nCancel operation complete\n";
                break;
           case 'T':
           {    cout << "\nCompute tax for how many recent sales: ";
                cin >> num;
                double tot = a.SalesTax(num);
                cout << "\nThe total tax is:  $ " << tot << '\n';
                break;
           }

           case 'M':   ShowMenu();     break;
           case 'X':   break;

           default:
                cout << "** Illegal menu option.  Try again...\n";
        }
   } while (choice != 'X');

   cout << "\nFinal balance for register # " << a.GetID() << " is $ "
        << a.GetAmount() << '\n';
}
```