Fall 2015.

Point Of Sale Application

OOP244 Assignment V2.0

Your job for this project is to prepare an application that manages the list of items stored in a store for sale. Your application keeps track of the quantity of items in the store, saved in a file and updates their quantity as they are sold.

The types of items kept in store are Perishable or Non-perishable.

- Perishables: Items that are mostly food and vegetable and have expiration date.
- Non-perishables: Items that are for household use and don't have expiry date.

To prepare the application you need to create several classes that encapsulate the different tasks at hand.

CLASSES TO BE DEVELOPED

The classes required by your application are:

Date A class that manages date and time.

PosIO A class that enforces iostream read and write functionality for the derived classes. An instance of any class derived from "PosIO" can read from or write to the console, or be saved to or retrieved from a text file.

Using this class the list of items can be saved to a file and retrieved later, and individual item specifications can be displayed on screen (in detail or as a bill item) or read from keyboard.

Item A class derived from PosIO, containing general information about an item in the store, like the name, Stock Keeping Unit (SKU) number, price, etc.

Perishable A class holding information for a Perishable item derived from the "Item" class that implements the requirements of the "PosIO" (i.e. implements the pure virtual methods of the PosIO class).

NonPerish A class derived from the "Item" class that implements the requirements of the "PosIO" class.

PosSys The class that manages Perishable and Non-Perishable items in a file. This class manages the listing, adding and updating the data file as the items are bought or sold in the store.

PROJECT DEVELOPMENT PROCESS

Your development work on this project has five milestones and therefore is divided into five deliverables. Shortly before the due date of each deliverable a tester program and a script will be provided to you. Use this tester program to test your solution and use the "submit" command for each of the deliverables as you do for your workshop.
Since the design of this project is an ongoing process, you may have to make minor changes to the previous milestones if there is a bug or incorrect design specs, when a new milestone is published.
The approximate schedule for deliverables is as follows

- Date class — Due: Kickoff (KO) + 5 days
- PosIO class — Due: KO + 7 days
- Item class — Due: KO + 13 days
- Perishable and NonPerish classes — Due: KO + 17 days
- PosSys class. — Due: KO + 23 days

FILE STRUCTURE FOR THE PROJECT

Each class will have its own header (.h) file and implementation (.cpp) file. The names of these files should be the class name.

In addition to the header files for each class, create a header file called "POS.h" that defines general values for the project, such as:

```
TAX (0.13)            The tax rate for the goods
MAX_SKU_LEN (7)       The maximum size of an SKU code

MIN_YEAR (2000)       The min year used to validate year input
MAX_YEAR (2030)       The max year used to validate year input

MAX_NO_ITEMS (2000)    The maximum number of records in the data file.
```

Include this header file wherever you use these values.

Enclose all the code developed for this application within the sict namespace.

Make sure all your header files are guarded against multiple inclusions by adding the following commands at the very beginning of your header file:

```
#ifndef SICT_HeaderFileName_H__
#define SICT_HeaderFileName_H__
```

And adding the following command to the very end of your header files:

```
#endif
```

The "HeaderFileName" in the first two commands are replaced with the name of your header file; for example if your header file name is PosSys.h then the commands will be:

```
#ifndef SICT_POSSYS_H__
#define SICT_POSSYS_H__
```

MILESTONE 1: THE DATE CLASS

The Date class encapsulates a single date and time value in the form of five integers: year, month , day, hour and minute. The date value is readable by an istream and printable by an ostream using the following format: YYYY/MM/DD, hh:mm or YYYY/MM/DD if the class it to hold only the date without the time. (if `_dateOnly` is true; see "`bool _dateOnly;`")

Complete the implementation of the Date class under the following specifications:

Member Data (attributes):

`int _year;` Year; a four digit integer between MIN_YEAR and MAX_YEAR, as defined in "POS.h"

`int _mon;` Month of the year, between 1 to 12

`int _day;` Day of the month, note that in a leap year February has 29 days, (see mday() member function)

`int _hour;` A two digit integer between 0 and 23 for the hour the a day.

`int _min;` A two digit integer between 0 and 59 for the minutes passed the hour

`int _readErrorCode;` Error code which identifies the validity of the date and, if erroneous, the part that is erroneous. Define the possible error values defined in the Date header-file as follows:

```
NO_ERROR   0  -- No error - the date is valid
CIN_FAILED 1  -- istream failed on accepting information
YEAR_ERROR 2  -- Year value is invalid
MON_ERROR  3  -- Month value is invalid
DAY_ERROR  4  -- Day value is invalid
HOUR_ERROR 5  -- Hour value is invalid
MIN_ERROR  6  -- Minute value is invalid
```

`bool _dateOnly;` A flag that is true if the object is to only hold the date and not the time.

Private Member functions (private methods):

`int value()const;` (this function is already implemented and provided)

This function returns a unique integer number based on the date-time. You can use this value to compare two dates. If the value() of one date-time is larger than the value of another date-time, then the former date-time (the first one) follows the second.

`void errCode(int errorCode);`

Sets the _readErrorCode member variable to one of the possible values listed above.

`void set(int year, int mon, int day, int hour, int min);` Sets the member variables to the corresponding arguments ant then sets the _readErrorCode to `NO_ERROR`.

Constructors:

No argument constructor: Sets the _dateOnly attribute to false and then sets the date and time to the current system's date and time using the set() function. (see "`void set();`")

Three argument constructor: This constructor sets the _dateOnly attribute to true and then accepts three integer arguments to set the values of _year, _mon and _day and sets _hour and _min to zero. It also sets the _readErrorCode to NO_ERROR.

Five argument constructor: Sets the _dateOnly attribute to false and then accepts five integer arguments to set the values of _year, _mon, _day, _hour and _min to zero. It also sets the _readErrorCode to NO_ERROR. The last argument of this constructor (int min) should have a default value of "0" so the constructor can be called with four arguments too.

Public member-functions (methods) and operators:

Relational operator overloads:

```
bool operator==(const Date& D)const;
bool operator!=(const Date& D)const;
bool operator<(const Date& D)const;
bool operator>(const Date& D)const;
bool operator<=(const Date& D)const;
bool operator>=(const Date& D)const;
```

These operators return the result of comparing the left operand to the right operand. These operators use the value() member function in their comparison. For example `operator<` returns true if `this->value()` is less than `D.value()`; otherwise returns false.

`int mdays()const;` (this function is already implemented and provided)
This function returns the number of days in the month based on _year and _mon values.

`void set();` (this function is already implemented and provided)
This function sets the date and time to the current date and time of the system.

Accessor or getter member functions (methods):

`int errCode()const;` Returns the _readErrorCode value.

`bool bad()const;` Returns true if _readErrorCode is not equal to zero.

`bool dateOnly()const;` Returns the _dateOnly attribute.

`void dateOnly(bool value);` Sets the _dateOnly attribute to the "value" argument. Also if the "value" is true, then it will set _hour and _min to zero.

IO member-funtions (methods):

`std::istream& read(std::istream& is = std::cin);`

Reads the date in the following format: YYYY/MM/DD (e.g. 2015/03/24) from the console if _date only is true or in the following format: YYYY/MM/DD, hh:mm (e.g. 2015/03/24, 22:15) if _dateonly is false. This function does not prompt the user. If the istream `(istr)` object fails at any point, this function sets `_readErrorCode` to `CIN_FAILED` and does NOT clear the istream object. If the istream `(istr)` object reads the numbers successfully, this function validates them. It checks that they are in range, in the order of year, month and day (see the general header-file and the mday() function for acceptable ranges for years and days respectively). If any number is not within range, this function sets `_readErrorCode` to the appropriate error code and omits any further validation. Irrespective of the result of the process, this function returns a reference to the istream `(istr)` object.

```
std::ostream& write(std::ostream& ostr = std::cout)const;
```

This function writes the date to the ostream `(ostr)` object in the following format: YYYY/MM/DD, if _dateOnly is true or YYYY/MM/DD, hh:mm if _dateOnly is false. Then it returns a reference to the ostream `(istr)` object.

Non-member IO operator overloads: (Helpers)

After implementing the Date class, overload the operator<< and operator>> to work with cout to print a Date, and cin to read a Date, respectively, from the console.

Use the read and write member functions. DO NOT use friends for these operator overloads.

Include the prototypes for these helper functions in the date header file.

Preliminary task

To kick-start the first milestone download the Visual Studio project, or individual files for milestone 1 from https://github.com/Seneca-OOP244/FP_MS1

MILESTONE 2: THE POSIO INTERFACE V1.0

The PosIO class is provided to enforce inherited classes to implement functions to work with **fstream** and **iostream** objects.

Download / Clone https://github.com/Seneca-OOP244/FP_MS2 and code /add a class called PosIO in PosIO.h file for your milestone 2 implementation:

You do not need the Date class for this milestone.

Pure virtual member functions (methods):

PosIO class, being an interface, only exists at class definition in a header file and has only four pure virtual member functions (methods) with following names:

1- save

Is a constant member function (does not modify the owner) and receives and returns references of std::fstream.

In future milestones children of PosIO will implement this method, when they are to be stored in a file.

2- load

Receives and returns references of std::fstream.

In future milestones children of PosIO will implement this method, when they are to be read from a file.

3- write

Is a constant member function and returns a reference of std::ostream.
write() receives two arguments: the first is a reference of std::ostream and the second is a bool argument called linear.

In future milestones children of PosIO will implement this method when they are to be printed on the screen in two different formats:
***Linear:** the class information is to be printed in one line*
***Form:** the class information is to be printed in several lines like a form.*

4- read

Returns and receives references of std::istream.

In future milestones children of PosIO will implement this method when their information is to be received from console.

As you already know, these functions only exist as prototypes in the class definition in the header file.

Submission:

Compile and test your PosIO.h with the provided class TestFile (TestFile.cpp, TestFile.h) and PosIOTester.cpp.

TestFile implements all the pure virtual methods of the PosIO to write and read, into and from a file and display the content of that file in linear or Form format.

When program runs for the first time, it will create a file call posfile.txt and asks you to add to its content by typing few lines from console.

Every time you run this program you will add to the content you added before. If everything compiles and works as described, then you can submit this milestone as usual:

$ ~professor.name/submit ms2 <ENTER>