Summer Semester 2018

Aid Management Application (AMA)

Version 3.4

When disaster hits a populated area, the most critical task is to provide immediately affected people with what they need as quickly and as efficiently as possible.

This project completes an application that manages the list of goods that need to be shipped to the disaster area. The client application tracks the quantity of items needed, tracks the quantity on hand, and stores the information in a file for future use.

The types of goods that need to be shipped are of two categories;

- Non-Perishable products, such as blankets and tents, which have no expiry date. We refer to products in this category as Product objects.
- Perishable products, such as food and medicine, that have an expiry date. We refer to products in this category as Perishable.

To complete this project you will need to create several classes that encapsulate your solution.

OVERVIEW OF THE CLASSES TO BE DEVELOPED

The classes used by the client application are:

Date

A class to be used to hold the expiry date of the perishable items.

ErrorState

A class to keep track of the error state of client code. Errors may occur during data entry and user interaction.

Product

A class for managing non-perishable products.

Perishable

A class for managing perishable products. This class inherits the structure of the "Product" class and manages an expiry date.

iProduct

An interface to the Product hierarchy. This interface exposes the features of the hierarchy available to the client application. Any "iProduct" class can

- read itself from or write itself to the console
- save itself to or load itself from a text file
- compare itself to a unique C-string identifier
- determine if it is greater than another product in the collating sequence
- report the total cost of the items on hand
- describe itself
- update the quantity of the items on hand
- report its quantity of the items on hand
- report the quantity of items needed
- accept a number of items

Using this class, the client application can

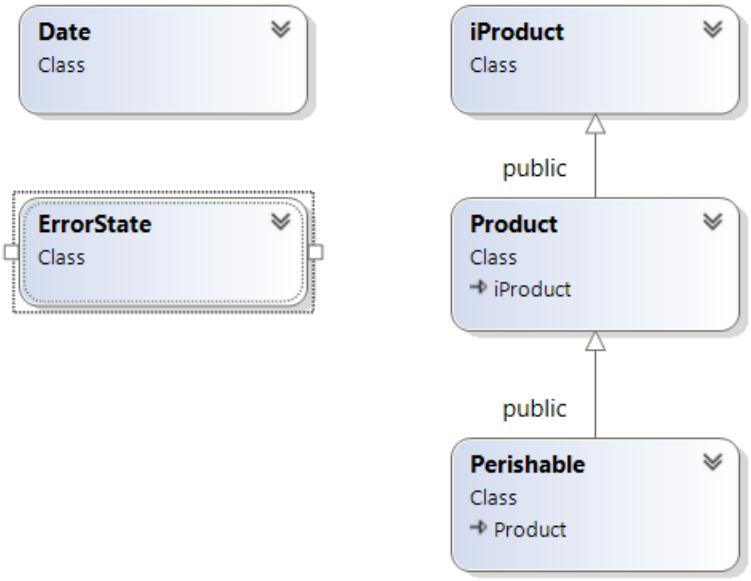
- save its set of iProducts to a file and retrieve that set later
- read individual item specifications from the keyboard and display them on the screen
- update information regarding the number of each product on hand

THE CLIENT APPLICATION

The client application manages the iProducts and provides the user with options to

- list the Products
- display details of a Product
- add a Product
- add items of a Product
- update the items of a Product
- delete a Product
- sort the set of Products

PROJECT CLASS DIAGRAM



PROJECT DEVELOPMENT PROCESS

The Development process of the project consists of 5 milestones and therefore 5 deliverables. Shortly before the due date of each deliverable a tester program and a script will be provided for testing and submitting the deliverable. The approximate schedule for deliverables is as follows

- Due Dates
  - The Date class — Due: July 13th, 11 days
  - The ErrorState class — Due: July 20th, 7 days
  - The Product class — Due: August 1st, 12 days
  - The iProduct interface — Due: August 3rd, 1 day
  - The Perishable class — Due: August 8th, 3 days

GENERAL PROJECT SUBMISSION

In order to earn credit for the whole project, you must complete all milestones and assemble them for the final submission.

Note that at the end of the semester you **MUST submit a fully functional project to pass this subject**. If you fail to do so, you will fail the subject. If you do not complete the final milestone by the end of the semester and your total average, without your project's mark, is above 50%, your professor may record an "INC" (incomplete mark) for the subject. With the release of your transcript you will receive a new due date for completion of your project. The maximum project mark that you will receive for completing the project after the original due date will be "49%" of the project mark allocated on the subject outline.

FILE STRUCTURE OF THE PROJECT

Each class has its own header (.h) file and its own implementation (.cpp) file. The name of each file is the name of its class.

Example: Class **Date** is defined in two files: **Date.h** and **Date.cpp**

All of the code developed for this application should be enclosed in the **AMA** namespace.

MILESTONE 4: THE IPRODUCT INTERFACE

The `iProduct` class is an interface that exposes the `Product` hierarchy to client applications. This class is abstract and cannot be instantiated. You will add and develop concrete classes of the hierarchy in the following milestone.

You do not need the `Date`, `ErrorState` or `Product` classes for this milestone.

Save your definition of the `iProduct` interface in a header file named **`iProduct.h`**.

The definition of your `iProduct` interface includes the following pure virtual member functions:

- `std::fstream& store(std::fstream& file, bool newLine=true) const`

  This query will receive a reference to an `fstream` object and an optional bool and return a reference to the `fstream` object. The bool argument will specify whether or not a newline should be inserted after each `iProduct` record. Implementations of this function will insert the `Product` records into the `fstream` object.

- `std::fstream& load(std::fstream& file)`

  This modifier will receive a reference to an `fstream` object and return a reference to that `fstream` object. Implementations of this function will extract `iProduct` records from the `fstream` object.

- `std::ostream& write(std::ostream& os, bool linear) const`

  This query will receive a reference to an `ostream` object and a `bool` and return a reference to the `ostream` object. The `bool` argument will specify whether or not the records should be listed on a single line or on separate lines. Implementations of this function will insert the `iProduct` record for the current object into the `ostream` object.

- `std::istream& read(std::istream& is)`

  This modifier will receive a reference to an `istream` object and returns a reference to the `istream` object. Implementations of this function will extract the `iProduct` record for the current object from the `istream` object.

- `bool operator==(const char*) const`

  This query will receive the address of an unmodifiable C-style null-terminated string and return true if the string is identical to the stock keeping unit of an `iProduct` record; false otherwise.

- `double total_cost() const`

This query will return the cost of a single unit of an `iProduct` with taxes included.

- `const char* name() const`

  This query will return the address of a C-style null-terminated string containing the name of an `iProduct`.

- `void quantity(int)`

  This modifier will receive an integer holding the number of units of an `iProduct` that are currently available. This function will set the number of units available.

- `int qtyNeeded() const`

  This query will return the number of units of an `iProduct` that are needed.

- `int quantity() const`

  This query will return the number of units of an `iProduct` that are currently available.

- `int operator+=(int)`

  This modifier will receive an integer identifying the number of units to be added to the `iProduct` and return the updated number of units currently available.

- `bool operator>(const iProduct&) const`

  This query will receive an unmodifiable reference to an `iProduct` object and return true if the current object is greater than the referenced `iProduct` object; false otherwise.

The following helper functions support your interface:

- `std::ostream& operator<<(std::ostream&, const iProduct&)`

  This helper function will receive a reference to an `ostream` object and an unmodifiable reference to an `iProduct` object and return a reference to the `ostream` object. Implementations of this function will insert the `iProduct` record for the referenced object into the `ostream` object.

- `std::istream& operator>>(std::istream&, iProduct&)`

  This helper function will receive a reference to an `istream` object and a reference to an `iProduct` object and return a reference to the `istream` object. Implementations of this function will extract the `iProduct` record for the referenced object from the `istream` object.

- `double operator+=(double&, const iProduct&)`

This helper function will receive a reference to a `double` and an unmodifiable reference to an `iProduct` object and return a `double`. Implementations of this function will add the total cost of the `iProduct` object to the `double` received and return the updated value of the `double`.

- **`iProduct* CreateProduct()`**

  This helper function will return the address of a `Product` object.

- **`iProduct* CreatePerishable()`**

  This helper function will return the address of a `Perishable` object.

Once you have defined this interface, compile the **`MyProduct.cpp`** and **`ms4_tester.cpp`** files provided. These two files should compile without error. The executable code should use your interface to append data fields to and read data fields from the **`ms4.txt`** file.

MILESTONE 4 SUBMISSION

If not on matrix already, upload the **`iProduct.h`**, **`MyProduct.h`**, **`MyProduct.cpp`** and **`ms4_tester.cpp`** files to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account: (replace profname.proflastname with your professor's Seneca userid)

```
~profname.proflastname/submit 200_ms4 <ENTER>
```

and follow the instructions.

Please note that a successful submission does not guarantee full credit for this workshop.

If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.