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

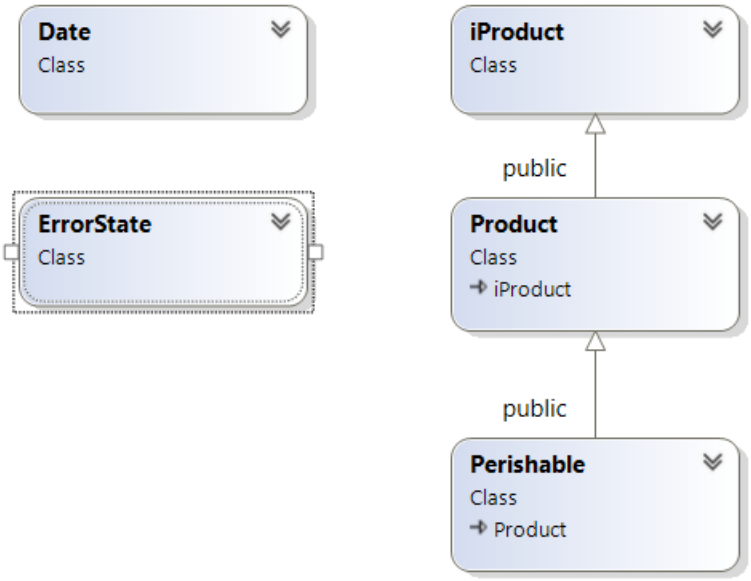
- save its set of iProducts to a file and retrieve that set later
- read individual item specifications from the keyboard and display them on the screen
- update information regarding the number of each product on hand

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 2: THE ERROR STATE CLASS

The `ErrorState` class manages the error state of client code and encapsulates the last error message.

Any client can define and store an `ErrorState` object. If a client encounters an error, it can set its `ErrorState` object to an appropriate message. The client sets the length of the message.

The `ErrorState` object reports whether or not an error has occurred. The **`isClear()`** query on the object reports if an error has occurred. If an error has occurred, the object can display the message associated with that error. The object can be send its message to an **`std::ostream`** object.

This milestone does not use the `Date` class.

Complete your implementation of the `ErrorState` class based on the following information:

Private member:

Data member:

A pointer that holds the address of the message, if any, stored in the current object.

Public member functions:

No/One Argument Constructor:

```
explicit ErrorState(const char* errorMessage = nullptr);
```

This function receives the address of a C-style null terminated string that holds an error message.

- If the address is **`nullptr`**, this function puts the object in a safe empty state.
- If the address points to a non-empty message, this function allocates memory for that message and copies the message into the allocated memory.

```
ErrorState(const ErrorState& em) = delete;
```

- A deleted copy constructor prevents copying of any `ErrorState` object.

```
ErrorState& operator=(const ErrorState& em) = delete;
```

- A deleted assignment operator prevents assignment of any `ErrorState` object to the current object.

```
virtual ~ErrorState();
```

- This function de-allocates any memory that has been dynamically allocated by the current object.

```
void clear();
```

- This function clears any message stored by the current object and initializes the object to a safe empty state.

```
bool isClear() const;
```

- This query reports returns true if the current object is in a safe empty state.

```
void message(const char* str);
```

This function stores a copy of the C-style string pointed to by **str**:

- de-allocates any memory allocated for a previously stored message
- allocates the dynamic memory needed to store a copy of **str** (remember to include 1 extra byte for the null terminator)
- copies the string at address **str** to the allocated memory.

```
const char* message()const;
```

- This query returns the address of the message stored in the current object.

Helper operator:

operator<<

- This operator sends an **ErrorState** message, if one exists, to an **std::ostream** object and returns a reference to the **std::ostream** object.
- If no message exists, this operator does not send anything to the **std::ostream** object and returns a reference to the **std::ostream** object.

Testing:

Test your code using the tester program supplied with this milestone.

MILESTONE 2 SUBMISSION

If not on matrix already, upload **ErrorState.h** and **ErrorState.cpp** with the tester 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 professors Seneca userid)

```
~profname.proflastname/submit 200_ms2 <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.