Fall Semester 2017

Aid Management Application (AMA)

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

This project prepares an application that manages the list of goods needed to be shipped to the area. The application should track the quantity of items needed, track the quantity on hand, and store the information in a file for future use.

The types of goods needed to be shipped are divided into two categories;

- Non-Perishable products, such as blankets and tents, which have no expiry date. We refer to products in this category as NonPerishable 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 the solution to this problem and to provide a solution for this application.

OVERVIEW OF CLASSES TO BE DEVELOPED

The classes used by this application are:

Date

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

ErrorMessage

A class to keep track of the errors occurring during data entry and user interaction.

Product

This interface (a class with “only” pure virtual functions) sets the requirements that derived classes must implement. These requirements have been set by the AidApp class. Any class derived from “Product” 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 application can

- save its set of Products 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

NonPerishable

A class for non-perishable products that implements the requirements of the "Product" interface (i.e. implements its pure virtual methods)

Perishable

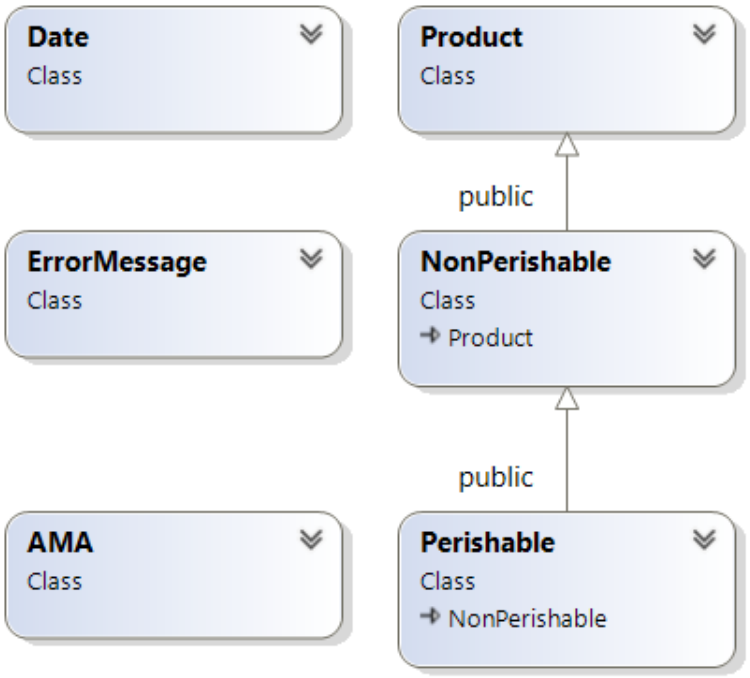
A class for perishable products that inherits from the "NonPerishable" class and provides an expiry date.

AidApp

The main application class manages the set of Products and provides the user with an interface 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 to you for testing and submitting the deliverable. The approximate schedule for deliverables is as follows

- Due Dates (Revised)
  - The Date class — Due: Nov 27th, 12 days
  - The ErrorMessage class — Due: Dec 4th, 7 days
  - The Product interface — Due: Dec 6th, 2 days
  - The NonPerishable class — Due: Dec 18th, 12 days
  - The Perishable class — Due: Dec 20th, 2 days
  - The AidApp class — Cancelled

GENERAL PROJECT SUBMISSION

In order to earn credit for the whole project, all milestones must be completed and assembled 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 the final milestone of your project is not completed by the end of the semester and your total average, without the 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 **`sict`** namespace.

MILESTONE 3: THE PRODUCT CLASS

The **Product** class is an interface that exposes the **Product** hierarchy to client applications. This class is abstract and cannot be instantiated. You will develop concrete classes that can be implemented in the following milestones. You do not need the **Date** or **ErrorMessage** class for this milestone.

Save your definition of the **Product** class in a header file named **Product.h**.

The definition of your **Product** class includes the following virtual member functions:

- `std::fstream& store(std::fstream& file, bool newLine=true) const` – a query that receives a reference to an `fstream` object and an optional bool and returns a reference to the `fstream` object. The bool argument specifies whether or not a newline is to be inserted after each `Product` record. Implementations of this function will insert the `Product` records into the `fstream` object.

- `std::fstream& load(std::fstream& file)` – a modifier that receives a reference to an `fstream` object and returns a reference to that `fstream` object. Implementations of this function will extract `Product` records from the `fstream` object.

- `std::ostream& write(std::ostream& os, bool linear) const` – a query that receives a reference to an `ostream` object and a `bool` and returns a reference to the `ostream` object. The `bool` argument specifies whether or not the records are to be listed on a single line or on separate lines. Implementations of this function will insert the `Product` records into the `ostream` object.

- `std::istream& read(std::istream& is)` – a modifier that receives a reference to an `istream` object and returns a reference to the `istream` object. Implementations of this function will extract the `Product` records from the `istream` object.

- `bool operator==(const char*) const` – a query that receives the address of an unmodifiable C-style string and returns true if the string is identical to the stock keeping unit of a `Product` record; false otherwise.

- `double total_cost() const` – a query that returns the cost of a single unit of the **Product** with taxes included.

- `const char* name() const` – a query that returns the address of a C-style string containing the name of the `Product`.

- `void quantity(int)` – a modifier that receives an integer holding the number of units of the `Product` that are available. This function sets the number of units available.

- `int qtyNeeded() const` – a query that returns the number of units of the `Product` that are needed.

- `int quantity() const` – a query returns the number of units of the `Product` that are available.

- `int operator+=(int)` – a modifier that receives an integer identifying the number of units to be added to the **`Product`** and returns the updated number of units available.

- `bool operator>(const Product&) const` – a query that receives an unmodifiable reference to a `Product` object and returns true if the current object is greater than the `Product` object; false otherwise.

The following helper functions support your interface:

- `std::ostream& operator<<(std::ostream&, const Product&)` – a helper that receives a reference to an `ostream` object and an unmodifiable reference to a `Product` object and returns a reference to the `ostream` object. Implementations of this function will insert a **`Product`** record into the `ostream`.

- `std::istream& operator>>(std::istream&, Product&)` – a helper that receives a reference to an `istream` object and an modifiable reference to a `Product` object and returns a reference to the `istream` object. Implementations of this function will extract the `Product` record from the `istream`.

- `double operator+=(double&, const Product&)` – a helper that receives a reference to a `double` and an unmodifiable reference to a `Product` object and returns a `double`. Implementations of this function will add the total cost of the `Product` object to the `double` received and return the updated `double`.

- `Product* CreateProduct()` – a helper that returns the address of a **`Product`**.

- `Product* CreatePerishable()` – a helper that returns the address of a perishable `Product`.

After you have defined this interface completely, compile the `MyProduct.cpp` file and the `ProductTester.cpp` provided. These two files should compile with no error, use your interface and read and append text to the `product.txt` file.

MILESTONE 3 SUBMISSION

If not on matrix already, upload **`Product.h`**, **`MyProduct.h`**, **`MyProduct.cpp`** and **`ProductTester.cpp`** 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 244_ms3 <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.