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

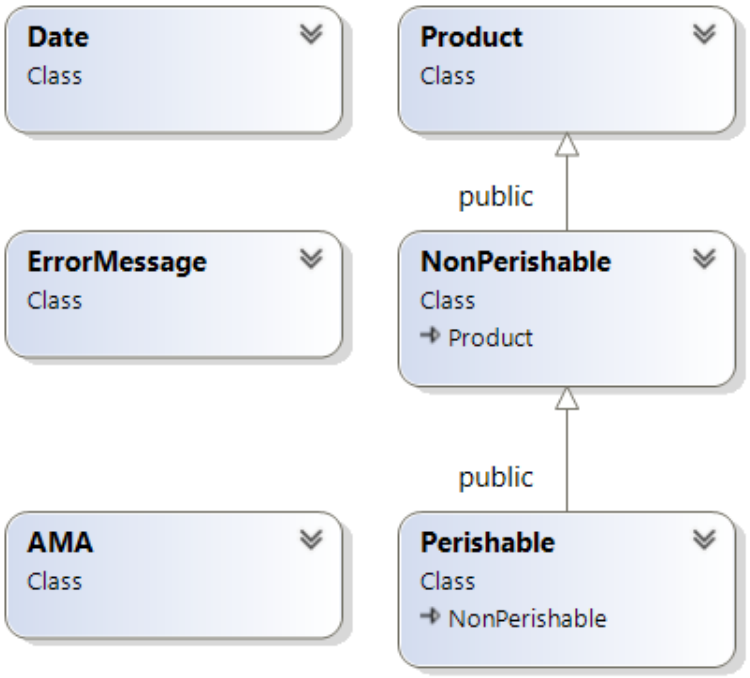
A class for perishable products that inherits from the "NonPerishable" class and provides an expiry date.

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 2: THE ERROR MESSAGE CLASS

The **ErrorMessage** class encapsulates an error message of client determined length and manages the error state of a client.

A client can create an **ErrorMessage** object and if the client encounters an error, it can set the object to an appropriate message. The object reports whether or not an error has occurred and can display the message. The **isClear()** query report if an error has occurred and the object can be send the error message to an **std::ostream**.

This milestone does not use the **Date** class.

Complete the implementation of the **ErrorMessage** class using following information:

Private member:

Data member:

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

Public member functions:

No/One Argument Constructor:

```
explicit ErrorMessage(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 it into the allocated memory.

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

A deleted copy constructor that prevents copying of an **ErrorMessage** object.

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

A deleted assignment operator that prevents assignment of an **ErrorMessage** object to the current object.

```
virtual ~ErrorMessage();
```

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 initialize 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 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 **ErrorMessage**, if a message exists, to an **std::ostream** object and returns a reference to the **std::ostream** object. If there is no message, this operator does not send anything 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 **ErrorMessage.h** and **ErrorMessage.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 244_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.