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

There are two categories for the types of goods that need to be shipped:

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

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 application are:

Date

A class that holds the expiry date of the perishable items.

Error

A class that tracks the error state of its client. Errors may occur during data entry and user interaction.

Good

A class that manages a non-perishable good object.

Perishable

A class that manages a perishable good object. This class inherits the structure of the “Good” class and manages a date.

iGood

An interface to the Good hierarchy. This interface exposes the features of the hierarchy available to the application. Any “iGood” class can

- read itself from the console or write itself to the console
- save itself to a text file or load itself from a text file
- compare itself to a unique C-style string identifier
- determine if it is greater than another good 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

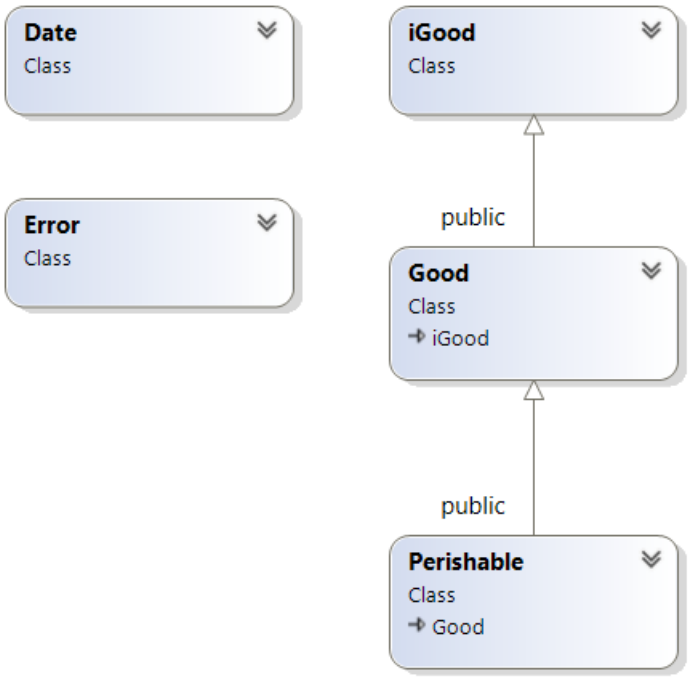
- save its set of iGoods to a file and retrieve that set at a later time
- read individual item specifications from the keyboard and display them on the screen
- update information regarding the number of each good on hand

THE CLIENT APPLICATION

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

- list the Goods
- display details of a Good
- add a Good
- add items of a Good
- update the items of a Good
- delete a Good
- sort the set of Goods

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 (at 11:59pm on each day)
  - The Date module — Due: November 2nd, 11 days
  - The Error module — Due: November 9th, 7 days
  - The Good module — Due: November 21st, 12 days
  - The iGood interface — Due: November 23rd, 2 days
  - The Perishable module — Due: November 28th, 3 days

SUBMISSION INSTRUCTIONS

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

Note that by the end of the semester you **MUST have submitted 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 belongs to its own module. Each module has its own header (.h) file and its own implementation (.cpp) file. The name of each file without the extension is the name of its class.

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

All the code developed for this application belongs to the **`aid`** namespace.

MILESTONE 2: THE ERROR MODULE

The **Error** class manages the error state of client code and encapsulates the last error message.

Any client can define and store an **Error** object. If a client encounters an error, the client can set its **Error** object to an appropriate message. The client specifies the length of the message.

The **Error** object reports whether or not any 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 **Error** 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 Error(const char* errorMessage = nullptr);
```

This constructor 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 into a safe empty state.
- If the address points to an empty message, this function puts the object into 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.

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

- A deleted copy constructor prevents copying of any **Error** object.

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

- A deleted assignment operator prevents assignment of any **Error** object to the current object.

```
virtual ~Error();
```

- 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
- if **`str`** points to a non-empty message, this function allocates the dynamic memory needed to store a copy of **`str`** (remember to include 1 extra byte for the null terminator) and copies the message into that memory
- if **`str`** points to an empty message, this function puts the current object in a safe empty state.

`const char* message() const;`

- if the current object is not in a safe empty state, this query returns the address of the message stored in the object
- if the current object is in a safe empty state, this query returns **`nullptr`**.

Helper operator:

`operator<<`

- This operator sends an **`Error`** 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 on Visual Studio.

MILESTONE 2 SUBMISSION

Upload **`Error.h`** and **`Error.cpp`** with the tester to your matrix account. Compile and rerun your code and make sure that everything works properly.

Then run the following command 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 milestone.

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