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.

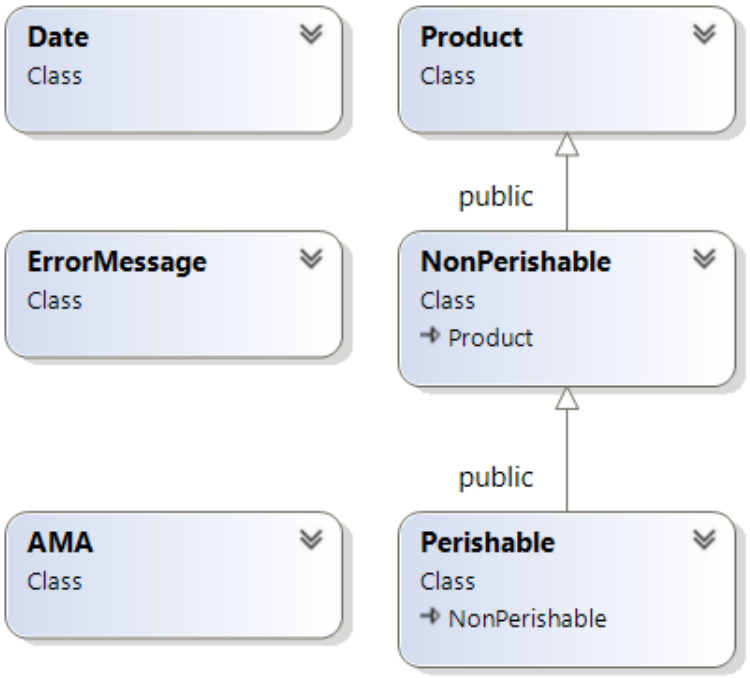
AidApp

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 5: THE PERISHABLE CLASS

Derive a class named **Perishable** from your **NonPerishable** class. Your **Perishable** class is a concrete class that encapsulates information for perishable products.

Define and implement your **Perishable** class in the **`sict`** namespace. Store your definition in a header file named **Perishable.h** and your implementation in a file named **Perishable.cpp**.

Your derived class uses a **Date** object. Your derived class does not need its own **ErrorMessage** object (the **NonPerishable** base class handles error processing).

Private data member:

A **Date** object holds the expiry date for the perishable product.

Public member functions:

Your design includes the following public member functions:

- `No argument Constructor` - this constructor passes the file record tag for a perishable product (`'P'`) to the base class constructor and sets the current object to a safe empty state.

- `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 modified `fstream` object. This function stores a file record for the current object. This function inserts into the `fstream` object the same information as the base class version inserts and then appends the expiry date to the file record.

  This function calls its base class version passing as arguments a reference to the `fstream` object and a false flag. This function then inserts a comma and the expiry date into the file record. If `newLine` is true, this function inserts a newline character (`'\n'`) before exiting. The file record written will look something like:

  ```
  P,1234,water,1.5,0,1,liter,5,2017/10/12<NEWLINE>
  ```

  If `newLine` is false, this function does not insert a newline character (`'\n'`) before exiting. The file record written will look something like:

  ```
  P,1234,water,1.5,0,1,liter,5,2017/10/12
  ```

Note that the first field in the file record is 'P', which was passed to the base class at construction time and inserted by the base class version of this function.

- `std::fstream& load(std::fstream& file)` – a modifier that receives a reference to an `fstream` object and returns a reference to that `fstream` object. This function extracts the fields for a single file record from the `fstream` object.

  This function calls its base class version passing as an argument a reference to the `fstream` object. This function then loads the expiry date from the file record using the `read()` function of the **Date** object. Then, this function extracts a single character 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` flag and returns a reference to the modified `ostream` object. The flag identifies the output format.

  This function calls its base class version passing as arguments a reference to the `ostream` object and the bool flag. If the current object is in an error or safe empty state, this function does nothing. If the current object is not in an error or safe empty state, this function inserts the expiry date into the `ostream` object. If `linear` is true, this function adds the expiry date on the same line for an outcome something like this:

  ```
  1234   |water               |   1.50|   1|liter     |   5|2017/10/12
  ```

  If `linear` is false, this function adds a new line character followed by the string "Expiry date: " and the expiry date for an outcome something like this:

  ```
  Sku: 1234
  Name: water
  Price: 1.50
  Price after tax: N/A
  Quantity On Hand: 1 liter
  Quantity Needed: 5
  Expiry date: 2017/10/12
  ```

This function does not insert a newline after the expiry date in the case of linear output (`linear` is true) or the case of line-by-line output (`linear` is false).

- `std::istream& read(std::istream& is)` – a modifier that receives a reference to an `istream` object and returns a reference to the modified `istream` object. This function populates the current object with the data extracted from the `istream` object.

  This function calls its base class version passing as its argument a reference to the `istream` object. If the base class object extracts data successfully, this function prompts for the expiry date and stores it in a temporary `Date` object. The prompt looks like with a single space after the colon:

  ```
  Expiry date (YYYY/MM/DD):
  ```

  If the temporary Date object is in an error state, this function stores the appropriate error message in the base class' error object and sets the state of the `istream` object to a failed state. The member function that sets it to a failed state is (`istream::setstate(std::ios::failbit)`). The messages that correspond to the error codes of a `Date` object are:

  CIN_FAILED: **Invalid Date Entry**
  YEAR_ERROR: **Invalid Year in Date Entry**
  MON_ERROR: **Invalid Month in Date Entry**
  DAY_ERROR: **Invalid Day in Date Entry**

  If the `istream` object is not in an error state, this function copy assigns the temporary `Date` object to the instance `Date` object. The member function that reports failure of an `istream` object is `istream::fail()`.

- `const Date& expiry() const` – this query returns the expiry date for the perishable product.

The following helper function supports your **`Perishable`** class:

- `Product* CreatePerishable()` – a helper that creates a **Perishable** object in dynamic memory and returns its address.

After you have implemented the **Perishable** class completely, compile your implementation files with the tester file provided. Your implementation files should compile with no error, use your interface and read and append text to the

`product.txt` and `productShortFile.txt` files. The output should look something like this:

```
--Product test:
----Taxed validation test:
Enter the following:
Sku: abc
Name: abc
Unit: abc
Taxed: a

 Sku: abc
 Name: abc
 Unit: abc
 Taxed? (y/n): a
Passed!
Message should be:  Only (Y)es or (N)o are acceptable
Your Error message: Only (Y)es or (N)o are acceptable
**** Press enter to continue ...

----Price validation test:
Enter the following:
Sku: abc
Name: abc
Unit: abc
Taxed: y
Price: abc

 Sku: abc
 Name: abc
 Unit: abc
 Taxed? (y/n): y
 Price: abc
Passed!
Message should be:  Invalid Price Entry
Your Error message: Invalid Price Entry
**** Press enter to continue ...

----Quantity validation test:
Enter the following:
Sku: abc
Name: abc
Unit: abc
Taxed: y
Price: 10
Quantity on hand: abc

 Sku: abc
 Name: abc
 Unit: abc
```

```
 Taxed? (y/n): y
 Price: 10
Quantity On hand: abc
Passed!
Message should be:  Invalid Quantity Entry
Your Error message: Invalid Quantity Entry
**** Press enter to continue ...

----Quantity Needed validation test:
Enter the following:
Sku: abc
Name: abc
Unit: abc
Taxed: y
Price: 10
Quantity on hand: 10
Quantity needed: abc

 Sku: abc
 Name: abc
 Unit: abc
 Taxed? (y/n): y
 Price: 10
Quantity On hand: 10
Quantity Needed: abc
Passed!
Message should be:  Invalid Quantity Needed Entry
Your Error message: Invalid Quantity Needed Entry
**** Press enter to continue ...

----Display test, the output of the Program and yours must match:
Enter the following:
Sku: 1234
Name: box
Unit: kg
Taxed: y
Price: 123.45
Quantity on hand: 1
Quantity needed: 5

 Sku: 1234
 Name: box
 Unit: kg
 Taxed? (y/n): y
 Price: 123.45
Quantity On hand: 1
Quantity Needed: 5
--Linear------------
Program: 1234   |box                | 139.50|   1|kg        |   5|
  Yours: 1234   |box                | 139.50|   1|kg        |   5|
--Form Display------
```

```
--Program:
Sku: 1234
Name : box
Price : 123.45
Price after tax : 139.50
Quantity On Hand : 1 kg
Quantity Needed : 5
--Yours:
Sku: 1234
Name: box
Price: 123.45
Price after tax: 139.50
Quantity On Hand: 1 kg
Quantity Needed: 5
**** Press enter to continue ...

----Storage and loading test, the output of the Program and yours must match:
--Store AmaProduct, program:
N,1234,box,123.45,1,1,kg,5
N,1234,box,123.45,1,1,kg,5
--Store Product, yours:
N,1234,box,123.45,1,1,kg,5
N,1234,box,123.45,1,1,kg,5
--Load Product:
Program: 1234   |box                | 139.50|   1|kg        |   5|
  Yours: 1234   |box                | 139.50|   1|kg        |   5|

--Perishable Item test:
----Expiry date Validation test:
Enter the following:
Sku: abc
Name: abc
Unit: abc
Taxed: n
Price: 10
Quantity on hand: 10
Quantity needed: 10
Expiry date: 10/1/1

 Sku: abc
 Name: abc
 Unit: abc
 Taxed? (y/n): n
 Price: 10
Quantity On hand: 10
Quantity Needed: 10
Expiry date (YYYY/MM/DD): 10/1/1
Passed!
Message should be:  Invalid Year in Date Entry
Your Error message: Invalid Year in Date Entry
**** Press enter to continue ...
```

```
----Display test, the output of the Program and yours must match:
Enter the following:
Sku: 1234
Name: water
Unit: liter
Taxed: n
Price: 1.5
Quantity on hand: 1
Quantity needed: 5
Expiry date: 2017/10/12

 Sku: 1234
 Name: water
 Unit: liter
 Taxed? (y/n): n
 Price: 1.5
Quantity On hand: 1
Quantity Needed: 5
Expiry date (YYYY/MM/DD): 2017/10/12
--Linear------------
Program: 1234   |water               |   1.50|   1|liter     |   5|2017/10/12
  Yours: 1234   |water               |   1.50|   1|liter     |   5|2017/10/12
--Form Display------
--Program:
Sku: 1234
Name : water
Price : 1.50
Price after tax : N/A
Quantity On Hand : 1 liter
Quantity Needed : 5
Expiry date : 2017/10/12
--Yours:
Sku: 1234
Name: water
Price: 1.50
Price after tax: N/A
Quantity On Hand: 1 liter
Quantity Needed: 5
Expiry date: 2017/10/12
**** Press enter to continue ...

----Storage and loading test, the output of the Program and yours must match:
--Store Perishable, program:
P,1234,water,1.5,0,1,liter,5,2017/10/12
P,1234,water,1.5,0,1,liter,5,2017/10/12
--Store Product, yours:
P,1234,water,1.5,0,1,liter,5,2017/10/12
P,1234,water,1.5,0,1,liter,5,2017/10/12
--Load Perishable:
Program: 1234   |water               |   1.50|   1|liter     |   5|2017/10/12
```

```
  Yours: 1234    |water                |    1.50|    1|liter      |    5|2017/10/12
```

MILESTONE 5 SUBMISSION

If not on matrix already, upload **Product.h, NonPerishable.h, NonPerishable.cpp, Perishable.h, Perishable.cpp, ErrorMessage.h, ErrorMessage.cpp, Date.h, Date.cpp** and 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_ms5 <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.