COMP 2404 -- Assignment #2

Due: Tuesday, February 24, 2015 at 9:00 PM

Collaboration: You may work in groups of no more than two (2) students

Goal

You will modify your BMS brig management system from Assignment #1 and separate persistent objects into storage using the Façade design pattern. You will also change the two collection classes to use dynamically allocated memory and grow every time an element (a pirate or a cell) is added to the collection.

Learning Objectives

- gain experience using a simple design pattern
- separate program logic for managing persistent objects
- practice with dynamically allocated memory in C++

Instructions:

1. Implement the collection classes

You will modify your collection classes (**`CellArray`** and **`PirateArray`**, or whatever name you gave them in Assignment #1) to be dynamically allocated arrays that grow every time an element is added. Each collection will contain two data members:

- a double pointer to the type of data contained in the array (cells or pirates); you will use this double pointer to hold the collection as an **array of data pointers**
- the current number of elements in the array

Notes:

- Do **not** allocate memory for the array of pointers until it is needed! The memory allocated for the array must be sufficient only for the current number of elements. Every time an element is added to the collection, you must allocate a new larger array and copy the content from the old array. Do not use C functions such as **`realloc`**; you must use **new** for all dynamically allocated memory.
- For extensibility and efficient design, the interfaces (i.e. the prototypes of the public member functions) of the two collection classes should be almost identical.
- All data (cells and pirates) must be dynamically allocated. There must exist **only one instance** of each cell and each pirate in the program. Do not make copies! Use multiple pointers to each instance instead.
- All instances of the collections are dynamically allocated. Make sure that you do not have memory leaks.
- In a future assignment, we will use a single collection class to hold either type of data, but we have not covered the appropriate material yet. You must keep the two collection classes separate for now.

2. Façade design pattern

The Façade design pattern is commonly used to provide a simplified interface, like a gateway, to a set of complex classes. With this pattern, the class users do not need to understand the more complex operations of each class they need, but instead they can invoke a simple operation on the Façade class which itself uses the more complex operations on the needed classes.

Note: The Façade class does not *replace* the complex classes nor does it implement their functionality.

For this assignment, you will implement a **`Storage`** class that will behave as a Façade class. We want the capability of data potentially being stored in different formats and locations, even somewhere else on a network! This means that we want to hide from the control, UI and entity objects where the data is stored, and how it is accessed. The **`Storage`** class will be the only class in the program that holds the persistent data, i.e. the permanent ("master") collection of cells and their individual collections of pirates.

The **`Storage`** class will provide two member functions:

- **`void retrieve(CellArray** cellArr)`** which allocates a new cell array and copies the entire contents of the persistent cell collection (and the pirate collections contained in each cell) into it
- **`void update(UpdateType action, CellArray* cellArr)`** which replaces the entire contents of the persistent cell collection with the contents of the given cell array. In a future assignment, you will be implementing the deletion of pirates. As a result, **`UpdateType`** must be an enumerated data type that indicates either an add or a delete action.

To assist with the copying of the persistent cell collection to and from **`Storage`**, you must to implement a copy constructor on the **`CellArray`** class, and your program must use it to copy the cell collection.

Remember: you are copying the collection, **not** the data!

Notes:

- There can only be one instance of the **`Storage`** class in your program. Do not make copies.
- Only the **`Storage`** class is allowed to maintain the persistent cell collection, no other classes are allowed to.
- All classes in the program must send queries and updates to the **`Storage`** class **every time** they need to display or modify the brig information.
- You **must** use the function prototypes as stated above.

Constraints

- your program must be written in C++, and **not** in C
- do not use any functions from the C standard library (e.g. printf, scanf, fgets, sscanf, malloc, free, string functions)
- do not use any classes or containers from the C++ standard template library (STL)
- do **not** use any global variables or any global functions other than **`main`**
- do **not** use structs; use classes instead
- objects should always be passed by reference, not by value

Submission

You will submit in cuLearn, before the due date and time, the following:

- a UML class diagram (as a PDF file) that corresponds to your program design
- one **`tar`** file that includes:
  - all source and header files for your brig management program
  - a Makefile
  - a readme file that includes:
    - a preamble (program author, purpose, list of source/header/data files)
    - compilation, launching and operating instructions