COMP 2404 -- Assignment #4

Due: Friday, March 21, 2014 at 11:00 AM

Goal

You will modify your movie database program from Assignment #3 and separate client and server functionality into two different processes. You will also implement some overloaded operators for the doubly linked list.

Learning Objectives

- adapt a repository-style program to a client-server architecture
- practice implementing overloaded operators on container objects

Instructions:

1. Overloaded operators

a. Implementing operators for the `List` class

You will implement the following overloaded operators for the **`List`** class:

- an assignment (=) operator that takes a reference to a second list as parameter and overwrites the **`List`** object with the elements from the second list
- an addition assignment (+=) operator that takes a data pointer as a parameter and adds it to the **`List`** object as a new element
- an addition assignment (+=) operator that takes a reference to a second list and adds each data element from that second list to the **`List`** object
- an addition (+) operator that takes a data pointer as a parameter and creates a new list comprised of all the elements of the **`List`** object as well as the new data element
- an addition (+) operator that takes a reference to a second list and creates a new list comprised of all the elements of the **`List`** object as well as all elements of the second list
- a subtraction assignment (-=) operator that takes a data pointer as a parameter and removes that element from the **`List`** object
- a subtraction assignment (-=) operator that takes a reference to a second list and removes each data element found in that second list from the **`List`** object
- a subtraction (-) operator that takes a data pointer as a parameter and creates a new list comprised of all the elements of the **`List`** object excluding the data element
- a subtraction (-) operator that takes a reference to a second list and creates a new list comprised of all the elements of the **`List`** object excluding all the elements of the second list

Notes:

- You must enable cascading everywhere appropriate
- You must reuse overloaded operators as much as possible

b. Using the overloaded operators

You will modify all your code to use the overloaded operators instead of the add/delete member functions.

Note:

- All the collections in your program must be **`List`** objects

2. Client-server architecture

You will separate your program into a server process and a client process. The server process will manage the persistent storage, potentially for multiple clients, while the client process contains the movie database logic and interacts with the user. The two processes will communicate using TCP/IP sockets, as provided in the **`Connection`** class found here. Figure 1 shows a high-level UML class diagram of how your code will be organized.

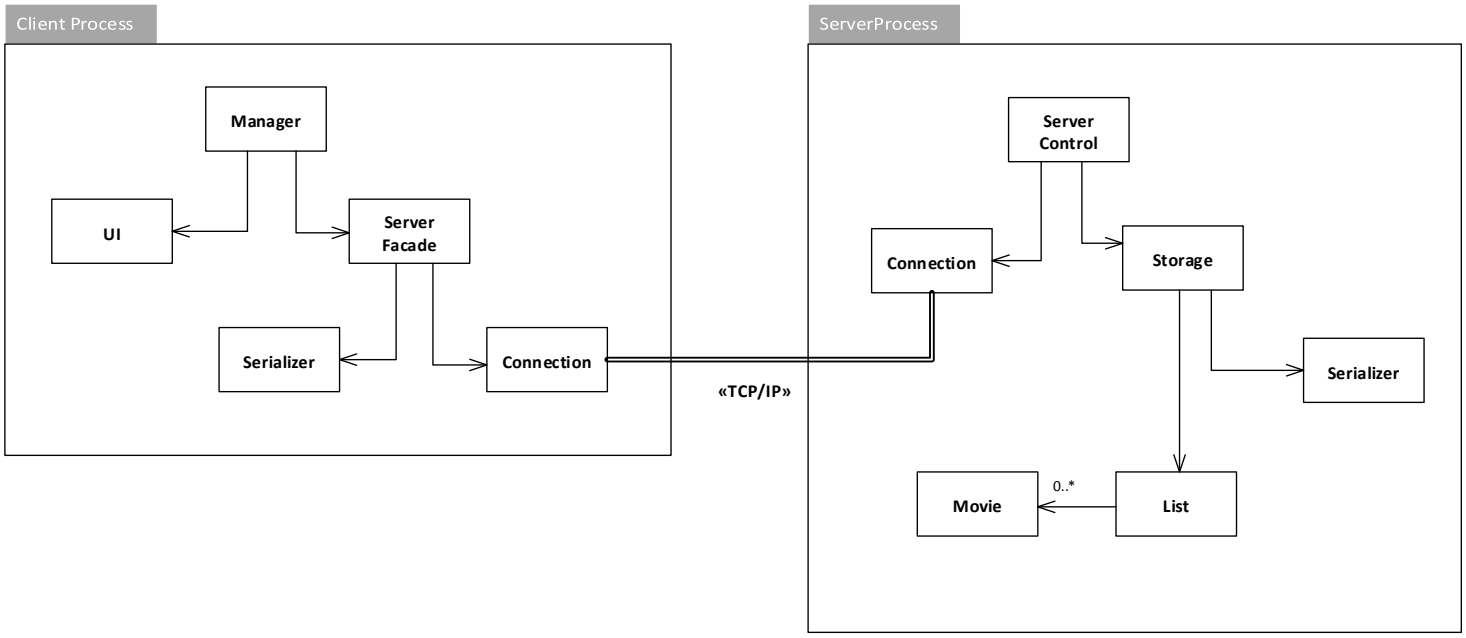

Figure 1 – High-level UML class diagram

a. Implement the server process

For the server process, you must:

- write a **`main`** function for the server process; all this function does is create a **`ServerControl`** object and invoke a launch operation on it
- implement a **`ServerControl`** class as the control class for the server process; the **`ServerControl`** object:
  - opens a server connection
  - waits for a connection request to come in from a client process
  - once a connection has been established, data is received from the client; if the first character of the client request is -1, it indicates that the client has terminated, so the server process must also terminate
  - passes the request to storage to be handled, and gets a response from storage
  - transmits the response back to the client process
  - goes back to waiting for another connection request to come in

Note: You have to read and thoroughly understand the **`Connection`** class in order to use it.

b. Modify the client process

For the client process, you must:

- use the same **`main`** function as in Assignments #1 to #3
- make no changes to the UI class, the **`Serializer`** class, or the **`Storage`** class
- make a small change to the control class, so that when the user selects 0 (Exit) from the main menu, the **`shutdown`** member function on the **`Server`** façade class is called in order to terminate the server process

- modify the **`Server`** façade class as follows:
  - remove the storage data member, since it will be accessible in the server process **only**
  - each time they are called, the **`update`** and **`retrieve`** member functions will open a client connection, transmit a serialized request over this connection, receive a serialized reply, deserialize the reply if necessary, and close the connection
  - add a **`shutdown`** member function that opens a client connection, sends a special request with -1 as the first character, and closes the connection

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 *detailed* 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 movie database program
  - a Makefile
  - a readme file that includes:
    - a preamble (program author, purpose, list of source/header/data files)
    - compilation, launching and operating instructions

Grading

- **Marking breakdown:**

Component	Marks
UML class diagram	10
Overloaded operators	30
Server process	35
Client process	25

- **Deductions:**
  - Up to 50 marks for any of the following:
    - the code does not compile using g++ in the VM provided for the course
    - the code cannot be tested because the program consistently crashes
    - the design does not generally follow correct design principles (e.g. data abstraction)
    - prohibited library classes or functions are used
  - Up to 20 marks for any of the following:
    - the Makefile or readme file is missing
    - global variables, global functions, or structs are used
    - objects are passed by value
    - the code is not correctly separated into header and source files
  - Up to 10 marks for missing comments or other bad style (non-standard indentation, etc.)
- **Bonus marks:**
  - Up to 5 extra marks are available for fun and creative additional features