HOMEWORK ASSIGNMENT #2

PROBLEM #1 (35 points)

There exist two inventory software servers S1 and S2 that maintain information about books in warehouses, i.e., they keep track of the number of books in warehouses. Books may be added or removed from the warehouses. Both servers support the following services:

Services supported by **server-S1**:

void AddBooks (ISBN string, warehouse int, n int)	//adds *n* books identified by ISBN to a warehouse
void DeleteBooks (ISBN string, warehouse int, n int)	//deletes *n* books identified by ISBN from a warehouse
int GetNumBooks (ISBN string)	//returns the total number of a book in all warehouses
int IsBook (ISBN string)	//returns 1, if a book exists; returns 0, otherwise

Services supported by **server-S2**:

void InsertBook (ISBM string, warehouse int)	//adds a book to a warehouse
void RemoveBook (ISBN string, warehouse int)	//deletes a book from a warehouse
int GetNumBooks (ISBN string)	//returns the total number of a book in warehouses
int IsBook (ISBN string)	//returns 1, if a book exists; returns 0, otherwise

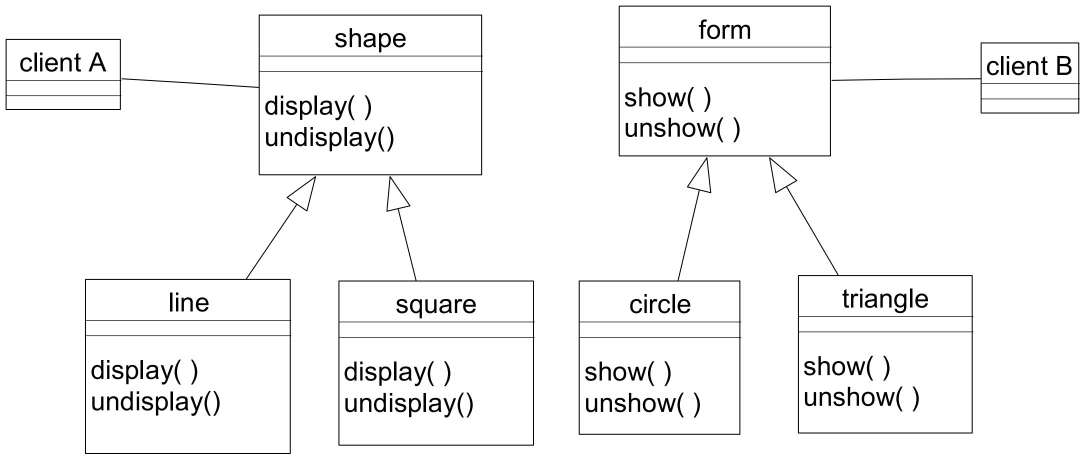
There exist two client processes and they request the following services:

Client-A	**Client-B**
void AddBooks (ISBN string, warehouse int, n int)	void InsertBook (ISBN string, warehouse int)
void RemoveBook (ISBN string, warehouse int)	void DeleteBooks (ISBN string, warehouse int, n int)
int GetNumBooks (ISBN string)	int GetNumBooks (ISBN string)
int IsBook (ISBN string)	int IsBook (ISBN string)

The client processes do not know the location (pointer) to servers that may provide these services. Devise a software architecture using a **Client-Broker-Server** architecture for this problem. In this design the client processes are not aware of the location of servers providing these services.

- Provide a class diagram for the proposed architecture. In your design, all components should be **decoupled** as much as possible.
- Provide the pseudocode for all operations of the following components/classes: (1) Broker, (2) Client Proxy of *Client-A*, (3) Server Proxy of *Server-2.*
- Provide a sequence diagram to show how *Client-A* gets *GetNumBooks(ISBN string)* service.

PROBLEM #2 (30 points)



A design of a system is shown above. In this system *clientA* uses objects of classes *line* and *square*. *clientB* uses objects of classes *circle* and *triangle.*

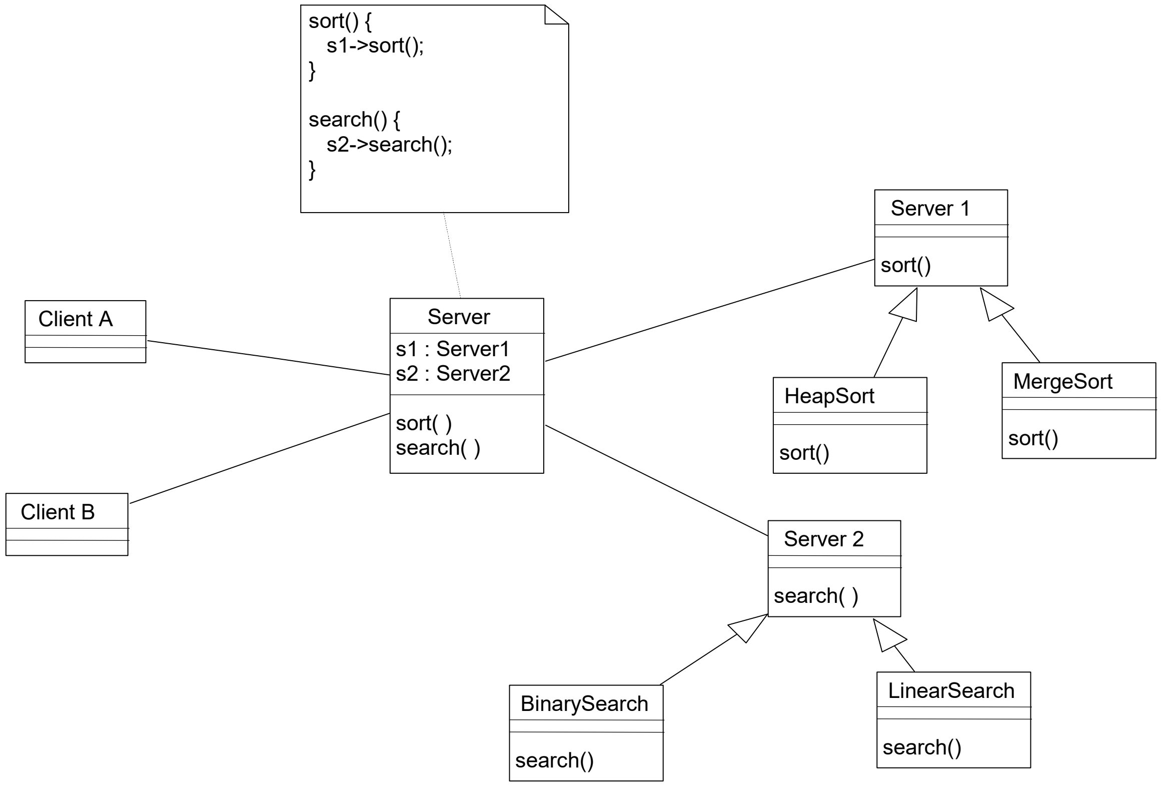
clientA would like to use objects (operations *show()* and *unshow()*) of classes *circle* and *triangle* by invoking operations *display()* and *undisplay()*. In addition, *clientB* would like to use objects (operations *display()* and *undisplay()*) of classes *line* and *square* by invoking operations *show()* and *unshow()*.

Provide a design with **minimal** modifications to the existing system using the **Adapter** design pattern in which (1) *clientA* is able to use objects of classes *circle* and *triangle* by invoking operations *display()* and *undisplay()*, and (2) *clientB* is able to use objects of classes *line* and *square* by invoking operations *show()* and *unshow().* Notice that none of the classes shown in the above class diagram should be modified. Provide two solutions that are based on:

1. an association-based version of the Adapter pattern
2. an inheritance-based version of the Adapter pattern

Provide a class diagram for each solution. You do not have to provide any description for classes/operations of the above class diagram (only new classes/operations should be described).

PROBLEM #3 (35 points)
Consider a design of the system shown below:



Client A and *Client B* get services, *sort()* and *search()*, directly from the *Server*. However, *Server* gets the appropriate services from *HeapSort* or *MergeSort* servers for *sort()* service. In addition, *Server* gets the appropriate services from *BinarySearch* or *LinearSearch* servers for *search()* service.

In this design clients do not have control where the services are coming from. However, *ClientA* by invoking *sort()* and *search()* on the *Server* would like to get *sort()* service from *HeapSort* server and *search()* service from *BinarySearch* server. On the other hand, *ClientB* by invoking *sort()* and *search()* on the *Server* would like to get *sort()* from *MergeSort* server and *search()* from *LinearSearch* server. Notice that the current design does not support this option.

Use the **abstract factory** design pattern to solve this problem. In your solution, the *Client* classes should be completely **de-coupled** from the issue of invoking services by the *Server* for appropriate versions of *sort()* and *search()*. Notice that none of the classes (and operations) shown in the above class diagram should be modified. However, new operations/classes can be introduced.

- Provide the class diagram and briefly describe the responsibility of each class and the functionality of each operation. In your design, all components should be **decoupled** as much as possible. You do not have to provide any description for classes/operations of the above class diagram (only new classes/operations should be described).
- Provide a sequence diagram to show how *ClientA* gets *sort() service* from *HeapSort* server and *search()* service from *BinarySearch* server by invoking *sort()* and *search()* on the *Server*.