Sample Exam #2

PROBLEM #1

There exist two servers **S1** and **S2**. Both servers support the following services:

Services supported by **server-S1**:
void Service1(string, int, int)
void Service2(string, int, int)
int Service3(string)
float Service4(string)

Services supported by **server-S2**:
void Service1(string, int)
void Service2(string, int)
int Service3(string)
float Service4(string)

There exist two client processes *ProcessA()* and *ProcessB()* and they request the following services:

Client-A	**Client-B**
void Service1(string, int, int)	void Service1(string, int)
void Service2(string, int)	void Service2(string, int, int)
int Service3(string)	int Service3(string)
float Service4(string)	float Service4(string)

The client processes do not know the location (pointer) to servers that may provide these services. Devise a software architecture using a **Client-Dispatcher-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 this architecture. Describe each component (class) of your design and operations supported by each class using the **pseudo-code**. However, you do not have to specify operations of *Server-1*, *Server-2* and *ClientA*. For *ClientB* only operation "*ProcessB()*" must be specified using pseudo-code showing how *ProcessB()* gets "*int Service3(string)"* service. In your design all components should be **decoupled** as much as possible.

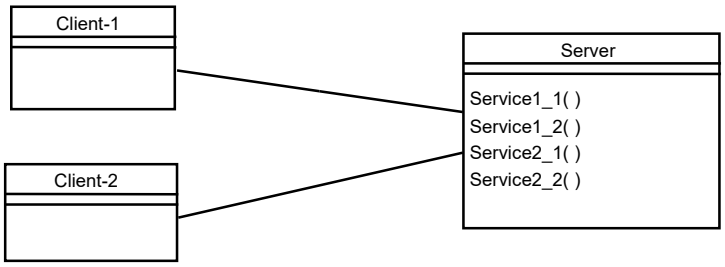
- Provide a sequence diagram to show how *ProcessB()* of *Client-B* gets "*int Service3(string)"* service.

PROBLEM #2

There exist two clients (*Client-1* and *Client-2*) and a *Server*. The server provides two services: *Service1()* and *Service2()*. There exist two versions of *Service1()*: *Service1_1()*, *Service1_2()*. In addition, there exist two versions of *Service2()*: *Service2_1()* and *Service2_2()*, where:

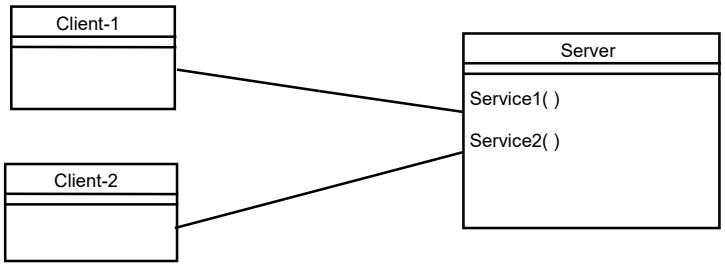
- *Client-1* invokes *Service1_1()* and *Service2_1()*
- *Client-2* invokes *Service1_2()* and *Service2_2()*

The current design is shown below:



In a better design Clients should be shielded from different versions of *Service1()* and *Service2()*. In the new design shown below:

- *Client-1* should invoke *Service1()* and *Service2()* to execute *Service1_1()* and *Service2_1()*.
- Similarly, *Client-2* should invoke *Service1()* and *Service2()* to execute *Service1_2()* and *Service2_2()*.

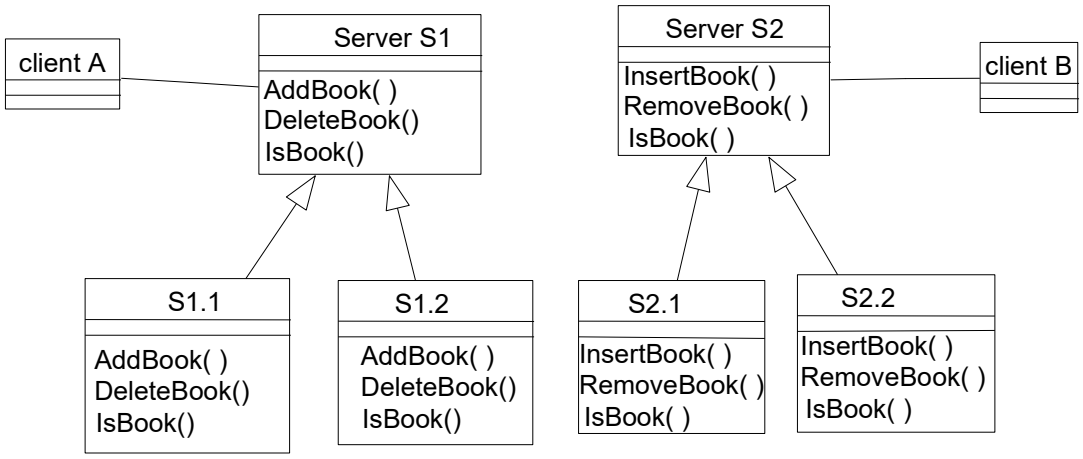


Use the **strategy pattern** and the **abstract factory** design pattern to solve this problem. In your solution the *Client* classes should be completely de-coupled from the issue of invoking appropriate versions of *Service1()* and *Service2()*.

- Provide the class diagram and describe the responsibility of each class and the functionality of each operation using pseudo-code. 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 showing how *Client-1* gets *Service1_1()* by invoking *Service1()* and then *Service2_1()* by invoking *Service2()* on the *Server*.

PROBLEM #3



A design of a system is shown above. In this system *clientA* invokes operations of servers *S1.1* and *S1.2* and *clientB* invokes operations of servers *S2.1* and *S2.2*.

clientA would like to invoke operations *InsertBook()*, *RemoveBook()* and *IsBook()* of servers *S2.1* and *S2.2* by invoking *AddBook()*, *DeleteBook()* and *IsBook()*. In addition, *clientB* would like to invoke operations *AddBook()*, *DeleteBook()* and *IsBook()* of servers *S1.1* and *S1.2* by invoking *InsertBook()*, *RemoveBook()* and *IsBook()*.

Provide a design with **minimal** modifications to the existing system using:

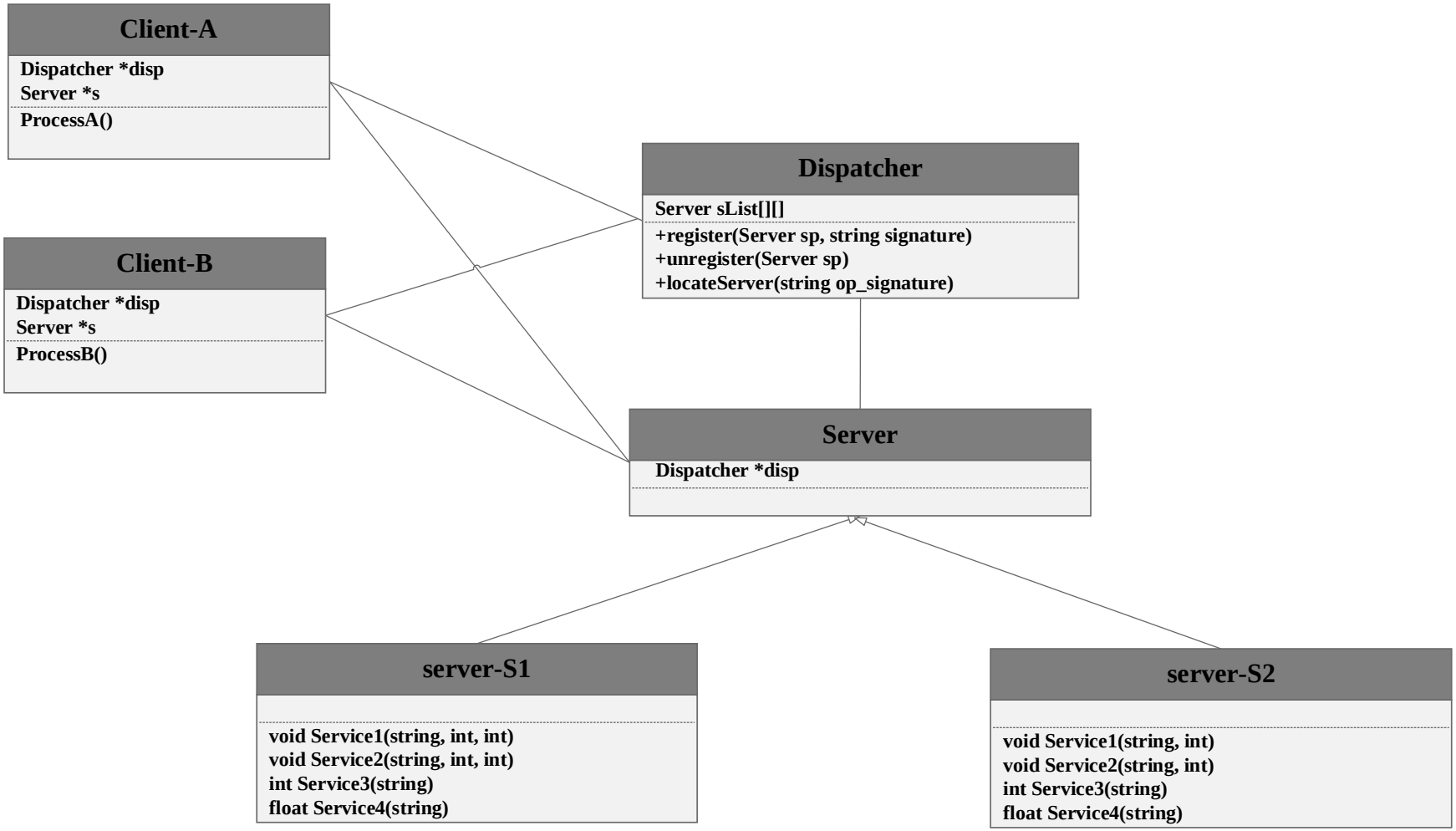
- **Association-Based Adapter** design pattern for *clientA*. As a result, *clientA* is be able to invoke operations of servers *S2.1* and *S2.2* by invoking *AddBook()*, *DeleteBook()* and *IsBook()*, and

- **Inheritance-Based Adapter** design pattern for *clientB.* As a result, *clientB* is to able to invoke operations of servers *S1.1* and *S1.2* by invoking *InsertBook()*, *RemoveBook()* and *IsBook()*.

Notice that none of the classes shown in the above class diagram should be modified. Provide a class diagram and briefly describe the responsibility of each class and the functionality of each operation using **pseudo-code**. You do not have to provide any description for classes/operations of the above class diagram (only new classes/operations should be described). In your design, all components should be **decoupled** as much as possible.

PROBLEM #1

Class Diagram:



Class Client-B

Dispatcher *disp

Server *s

Operations

```
void ProcessB() {
        s =  disp->locateServer("int Service3(string)");
        int result = s->Service3("abc") ;
}
```

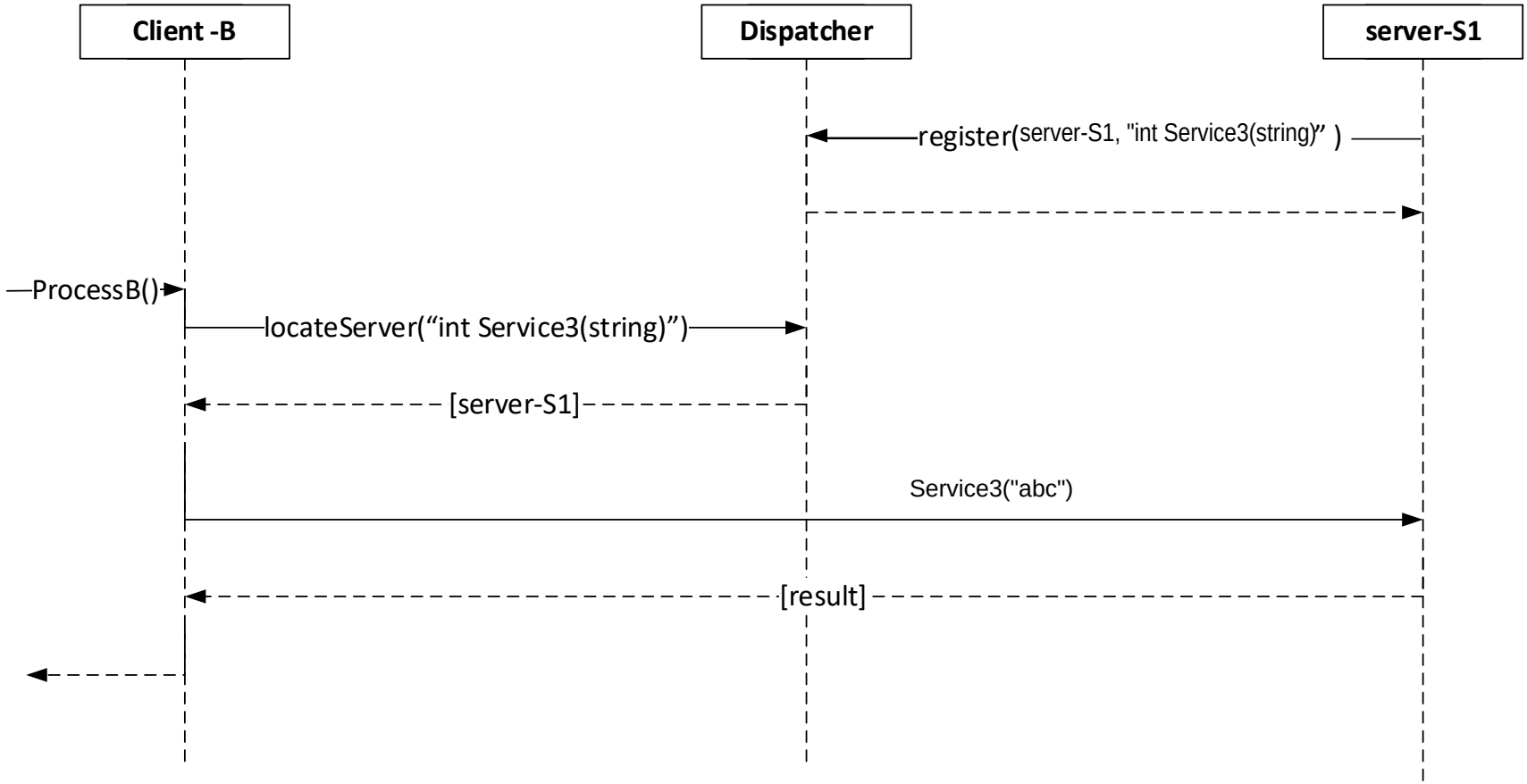
Class Dispatcher

Server sList[][]

Operations

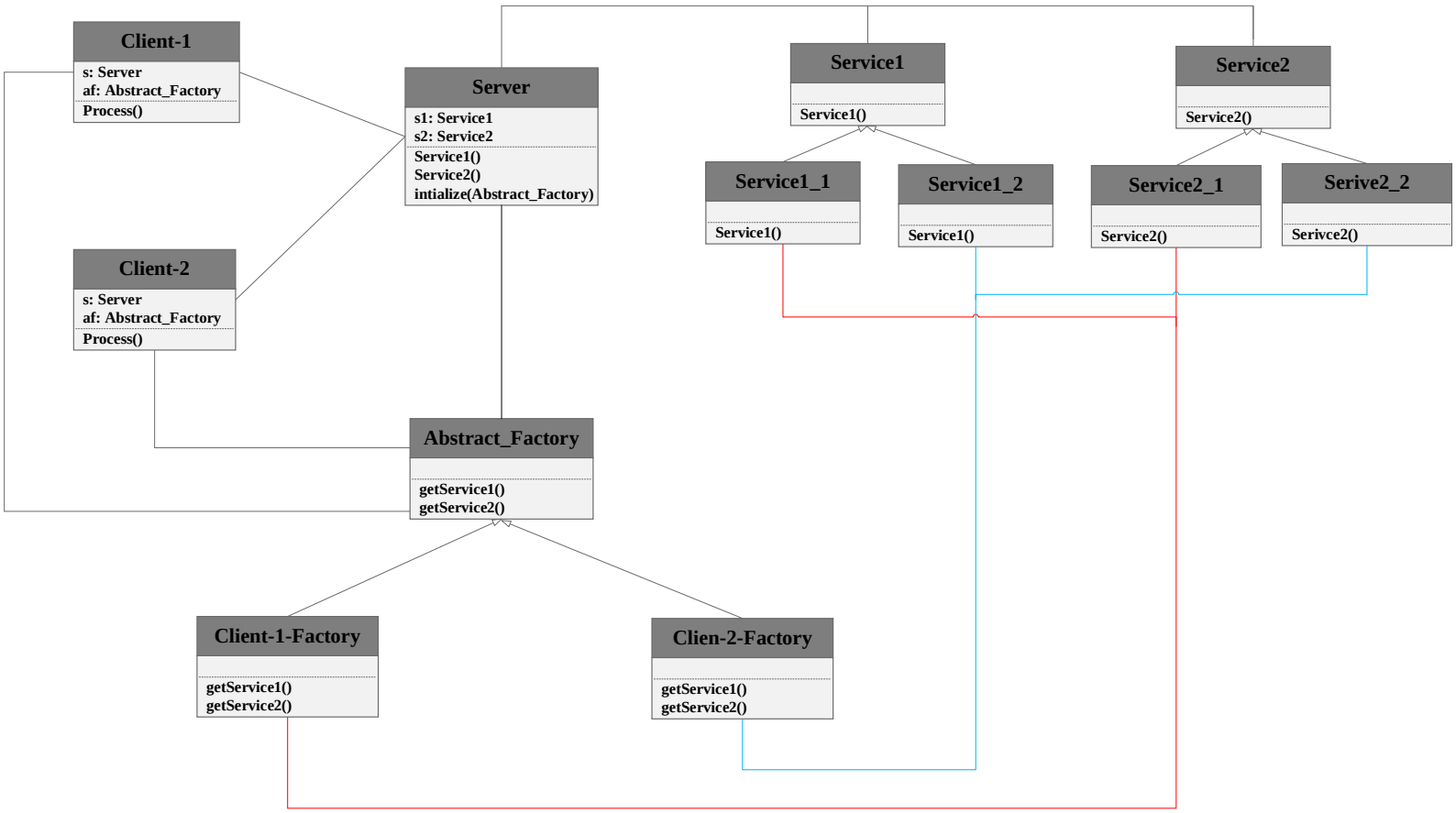
```
register(Server s, String signature){
        add s to sList
        add signature to sList[s]
}

unregister(Server s){
        remove s from sList
}

locateServer(string op_signature){
        for every server in sList
            IF sList[server] contains op_signature THEN
                return server
            ENDIF
}
```

Sequence Diagram



PROBLEM #2

Class Diagram: `



Class Cleint-1-Factory

```
getService1(){
    return new Service1_1();
}

getService2(){
    return new Service2_1();
}
```

Class Cleint-2-Factory

```
getService1(){
    return new Service1_2();
}

getService2(){
    return new Service2_2();
}
```

Class Server

```
s1: Service1
s2: Service2

Service1(){
    s1->Service1();
}

Service2(){
    s2-> Service2();

initialize(Abstract_Factory *af) {
    s1 =  af->getService1()
    s2 =  af->getService2()
}
```

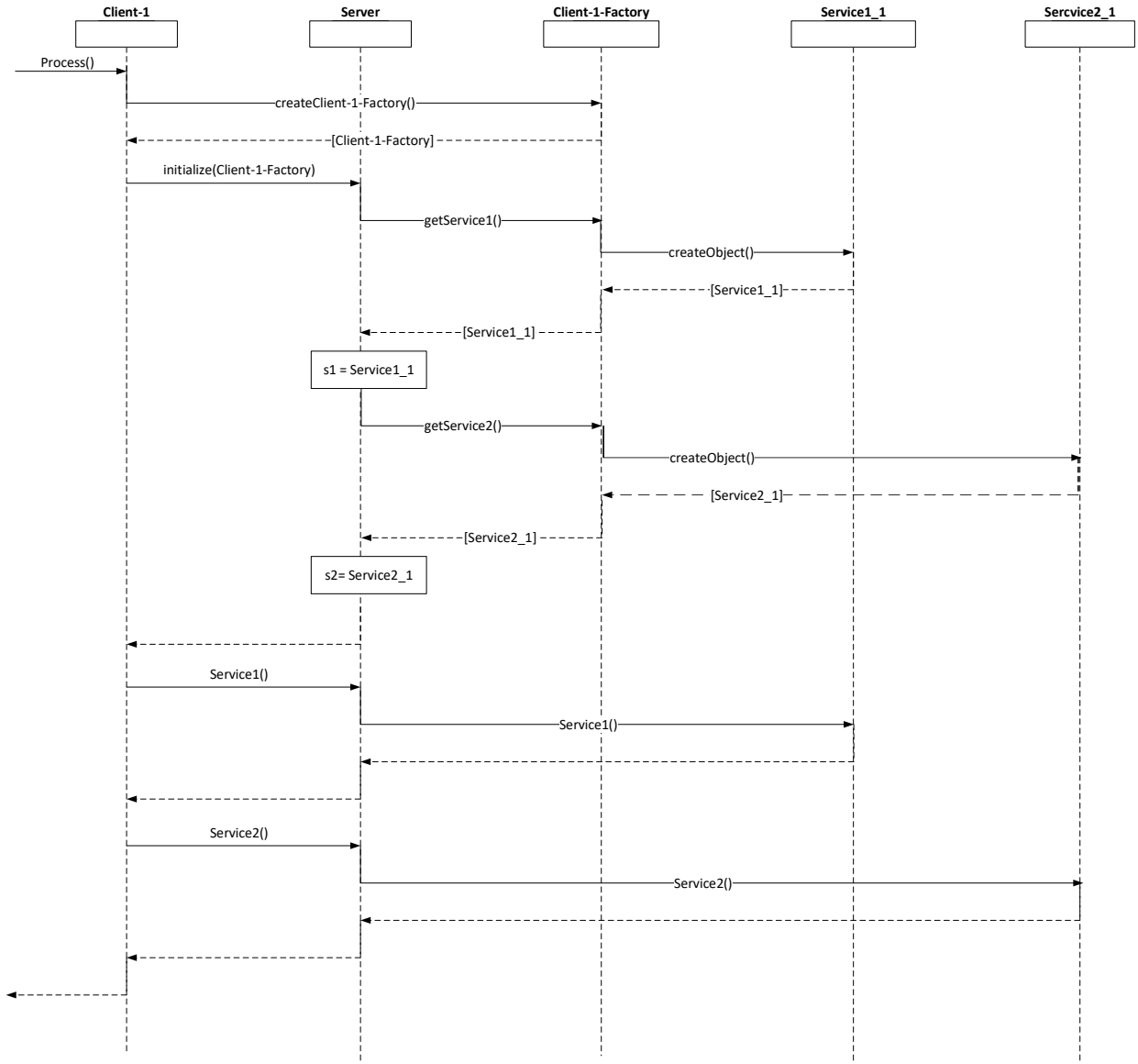
Class Client1

```
s: Server
af: Abstract_Factory

Process() {
    af= Client-1-Factory() ; // Clientl-1-Factory()
method creates an Client-1-Factory object and
returns its address
    s ->initialize(af);
    s ->Service1()
    s ->Service2()
}
```

Class Client2

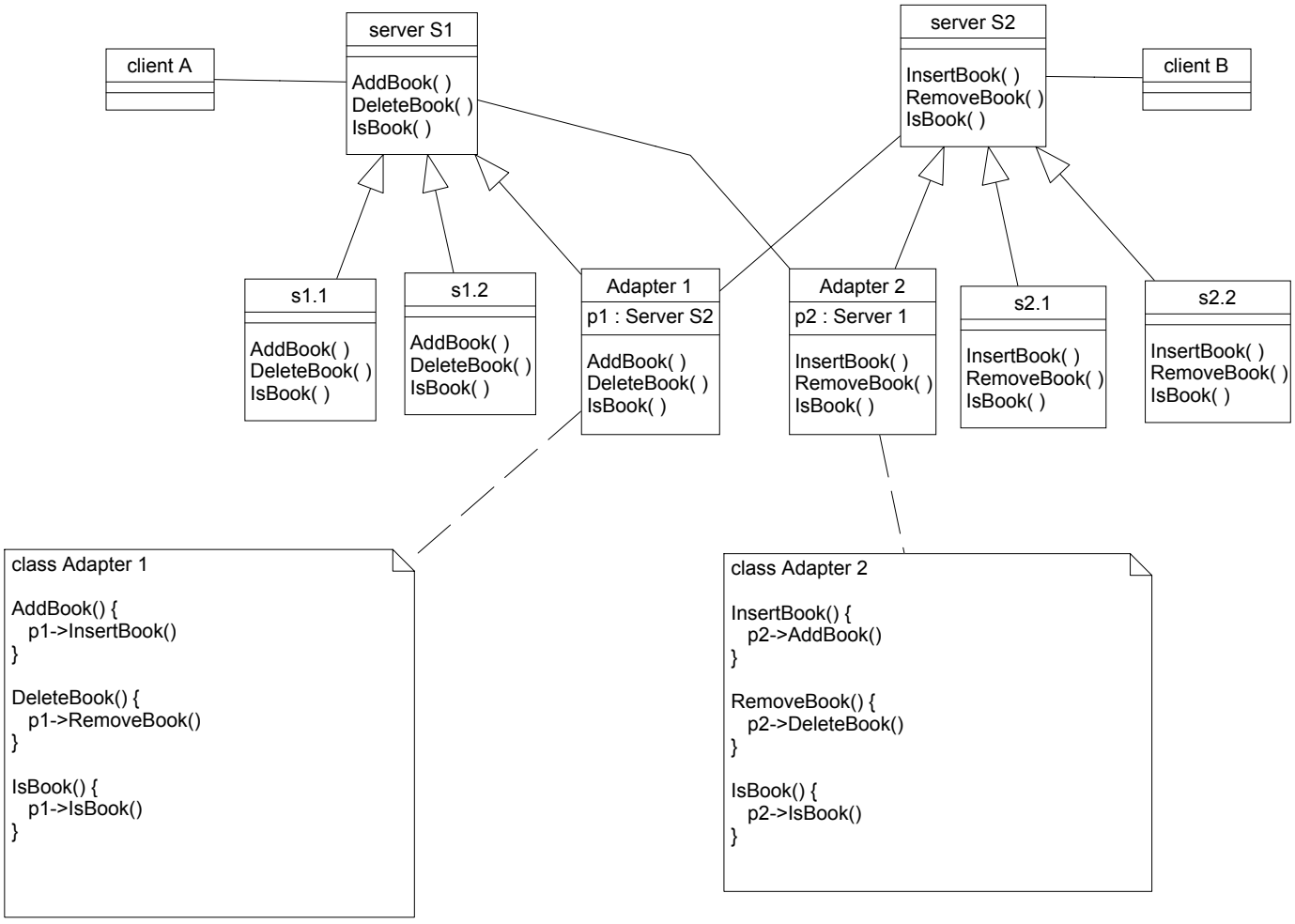
```
s: Server
af: Abstract_Factory

Process() {
    af= Client-2-Factory() ; // Clientl-2-Factory()
method creates an Client-2-Factory object and
returns its address
    s ->initialize(af);
    s ->Service1()
    s ->Service2()
}
```

Sequence Diagram:



PROBLEM #3

: Association-based version of the Adapter pattern



: Inheritance-based version of the Adapter pattern