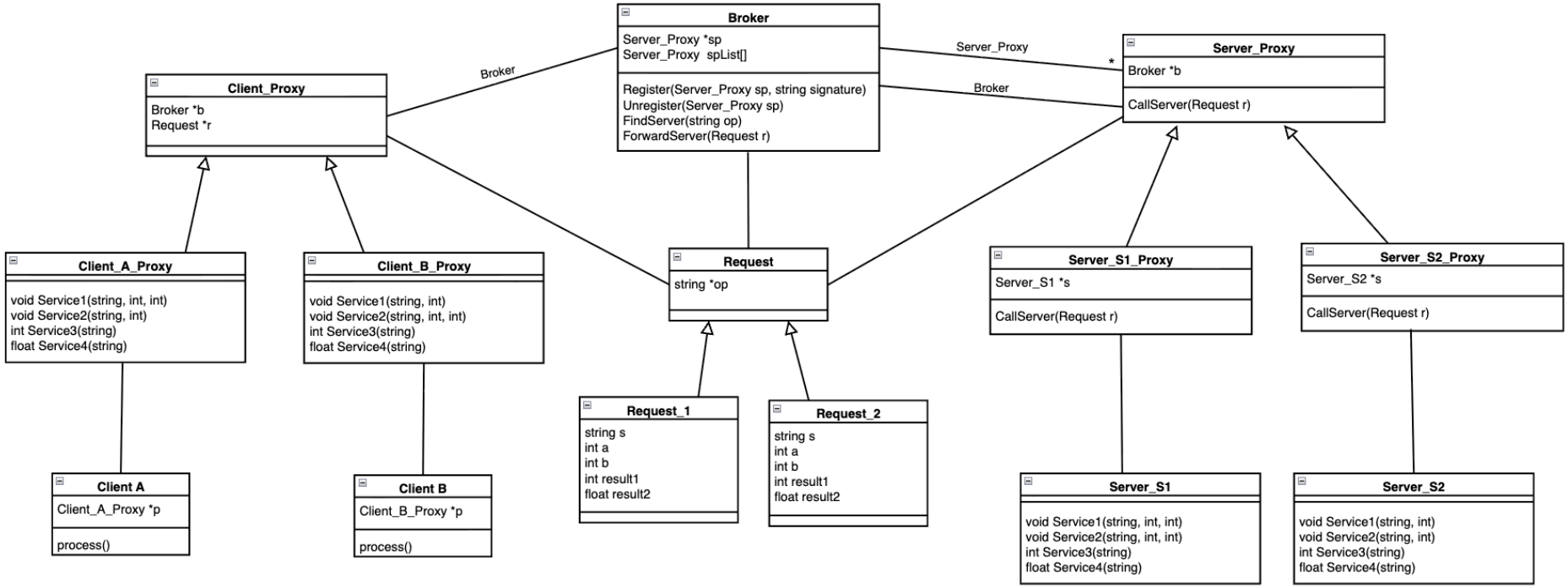
Problem 1: Class Diagram



Problem 1: Pseudo-code

Class Broker {
Server_Proxy *sp
Server_Proxy spList[]

Register(Server_Proxy sp, string signature) {
Add sp with the signature to the spList[]
}

Unregister(Server_Proxy sp) {
Remove sp from the spListp[]
}

FindServer(string op) {
For every sp in SpList[]
    If spList[sp] contains op then
        Return sp
    Else
        Return Null
    EndIf
}

ForwardServer(Request r) {
sp = FindServer(r.op)
If (sp != Null)
    sp=> CallServer(r)
Else
//Server not found
EndIf
}
}

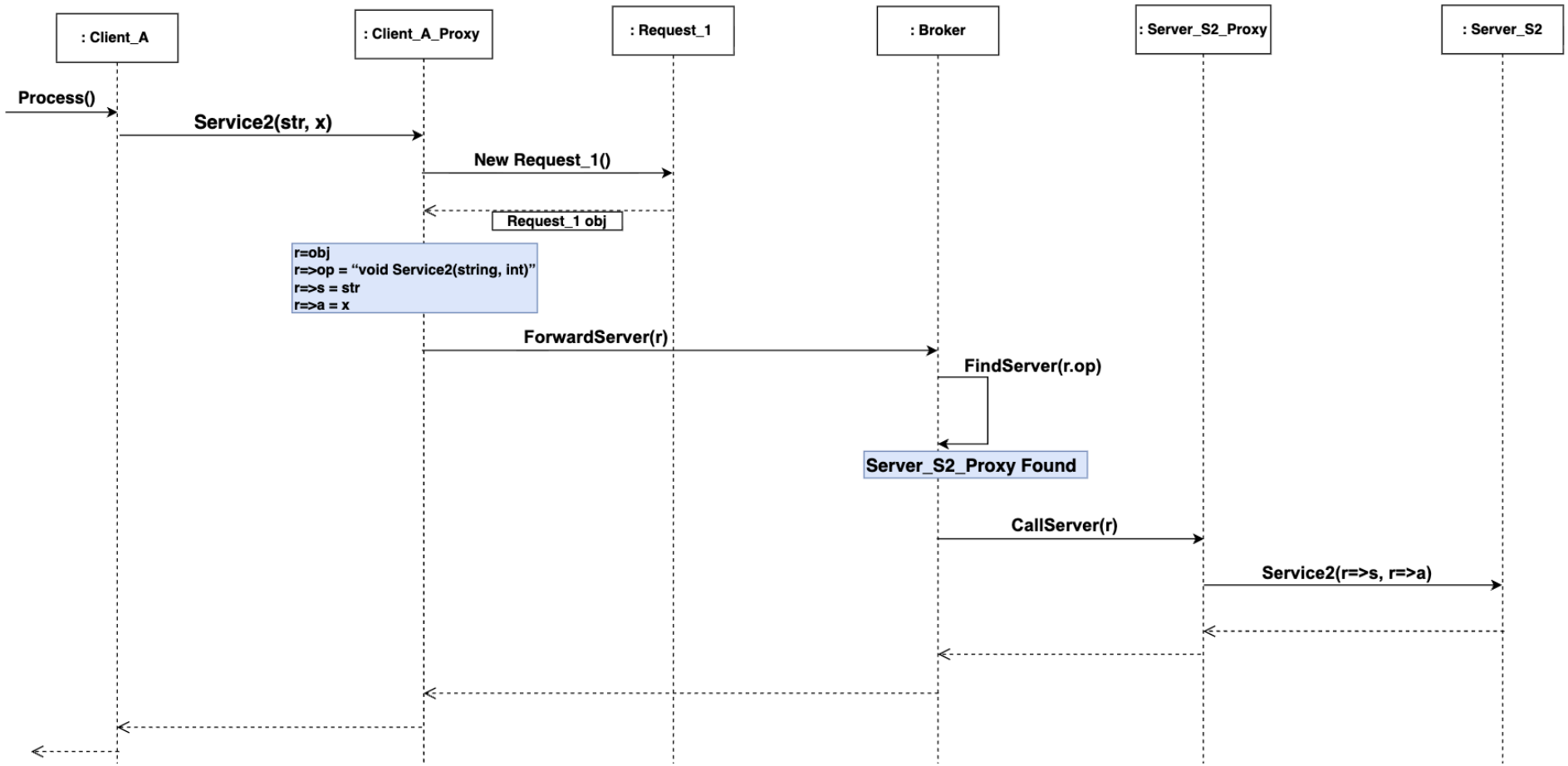
Class Client_A_Proxy {
b // Broker pointer inherited from Client_Proxy Class

void Service1(string str, int x, int y) {
Request r = new Request_1()
r=>op = “void Service1(string, int, int)”
r=>s = str
r=>a = x
r=>b = y
b=> ForwardServer(r)
}

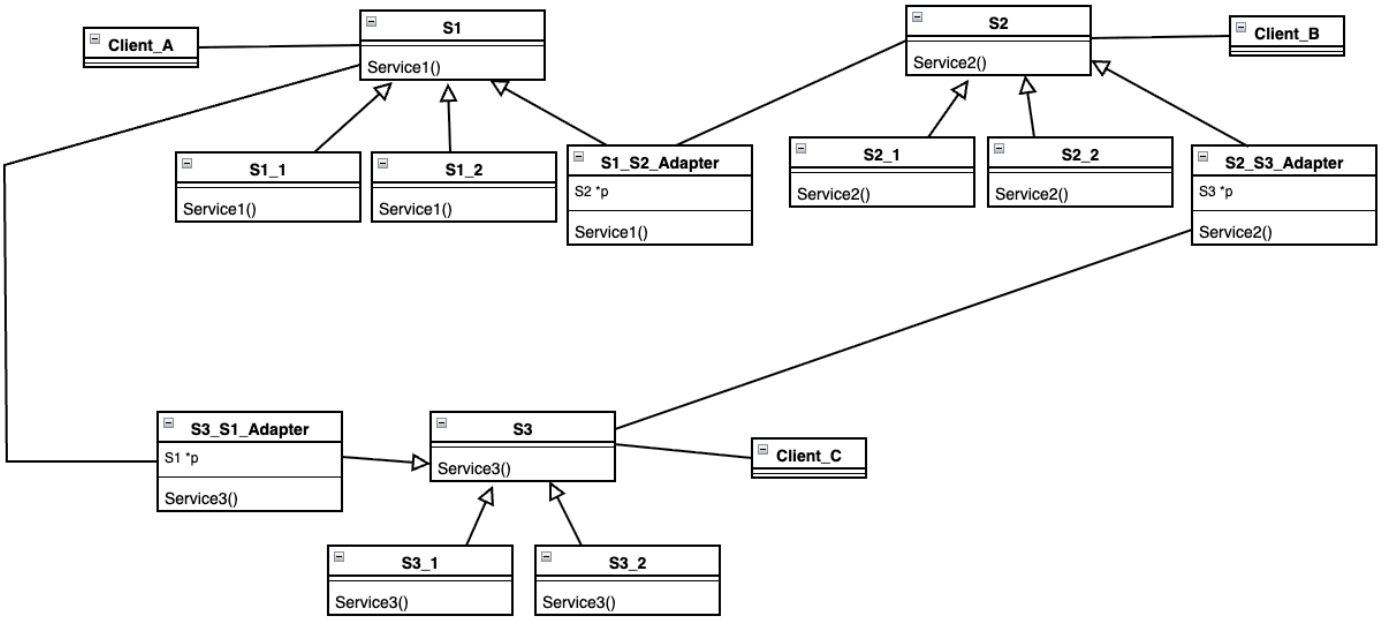
```
void Service2(string str, int x) {
Request r = new Request_1()
r=>op = “void Service2(string, int)”
r=>s = str
r=>a = x
b=> ForwardServer(r)
}


int Service3(string str) {
Request r = new Request_1()
r=>op = “int Service3(string)”
r=>s = str
b=> ForwardServer(r)
Return r=>result1
}


float Service4(string str) {
Request r = new Request_1()
r=>op = “float Service4(string)”
r=>s = str
b=> ForwardServer(r)
Return r=>result2
}
}
```

```
Class Server_S2_Proxy {
Server_S2 *s

CallServer(Request r){

If (r=>op == “void Service1(string, int)”)
        s=> Service1(r=>s, r=>a)
Else If (r=>op == “void Service2(string, int)”)
        s=> Service2(r=>s, r=>a)
Else If (r=>op == “int Service3(string)”)
        r=> result1= s=>Service3(r=>s)
Else If (r=>op == “float Service4(string)”)
        r=> result2= s=>Service4(r=>s)
}
}
```

Problem 1: Sequence Diagram

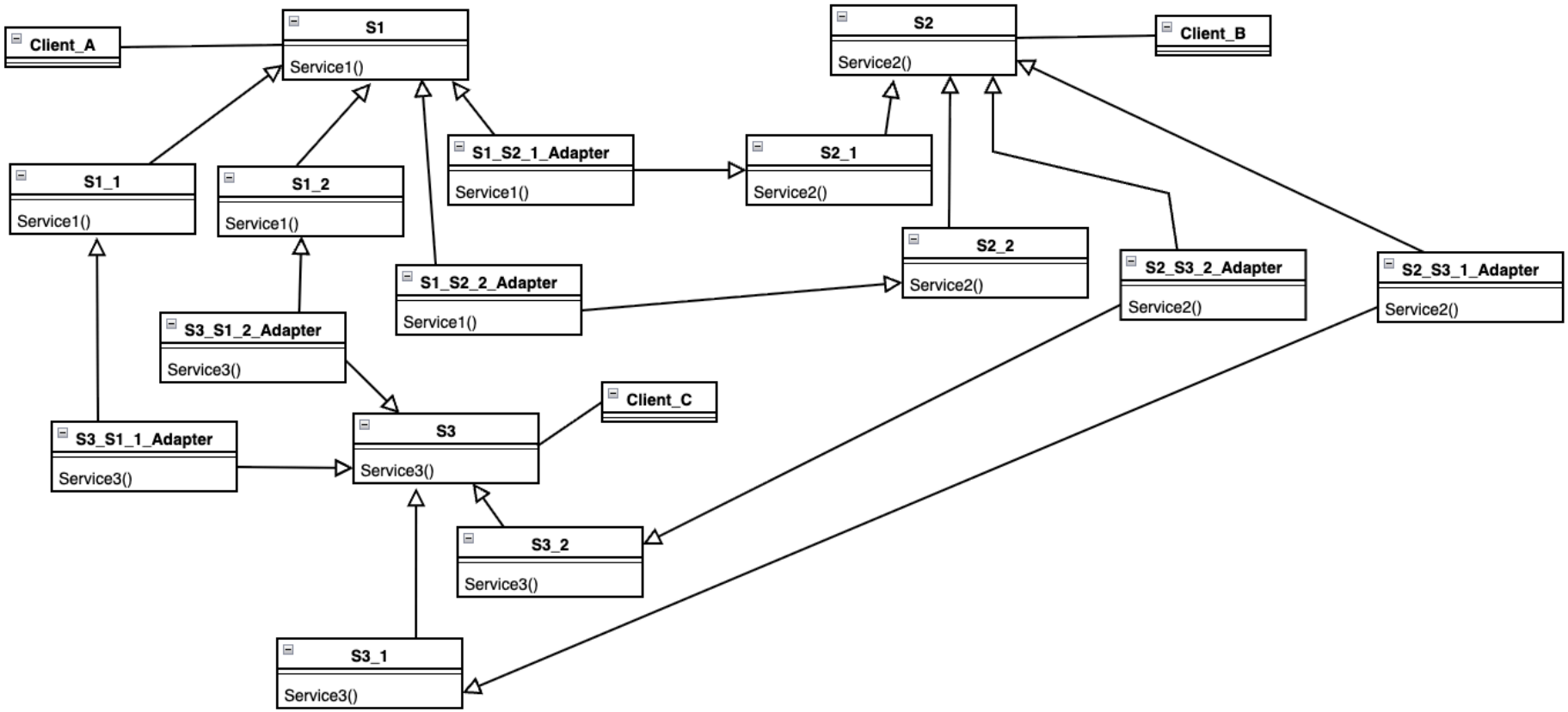


Problem #2
Association-based




```
Class S1_S2_Adapter{
S2 *p
Service1(){p => Service2()}
}

Class S2_S3_Adapter{
S3 *p
Service2(){p => Service3()}
}

Class S3_S1_Adapter{
S1 *p
Service3(){p => Service1()}
}
```

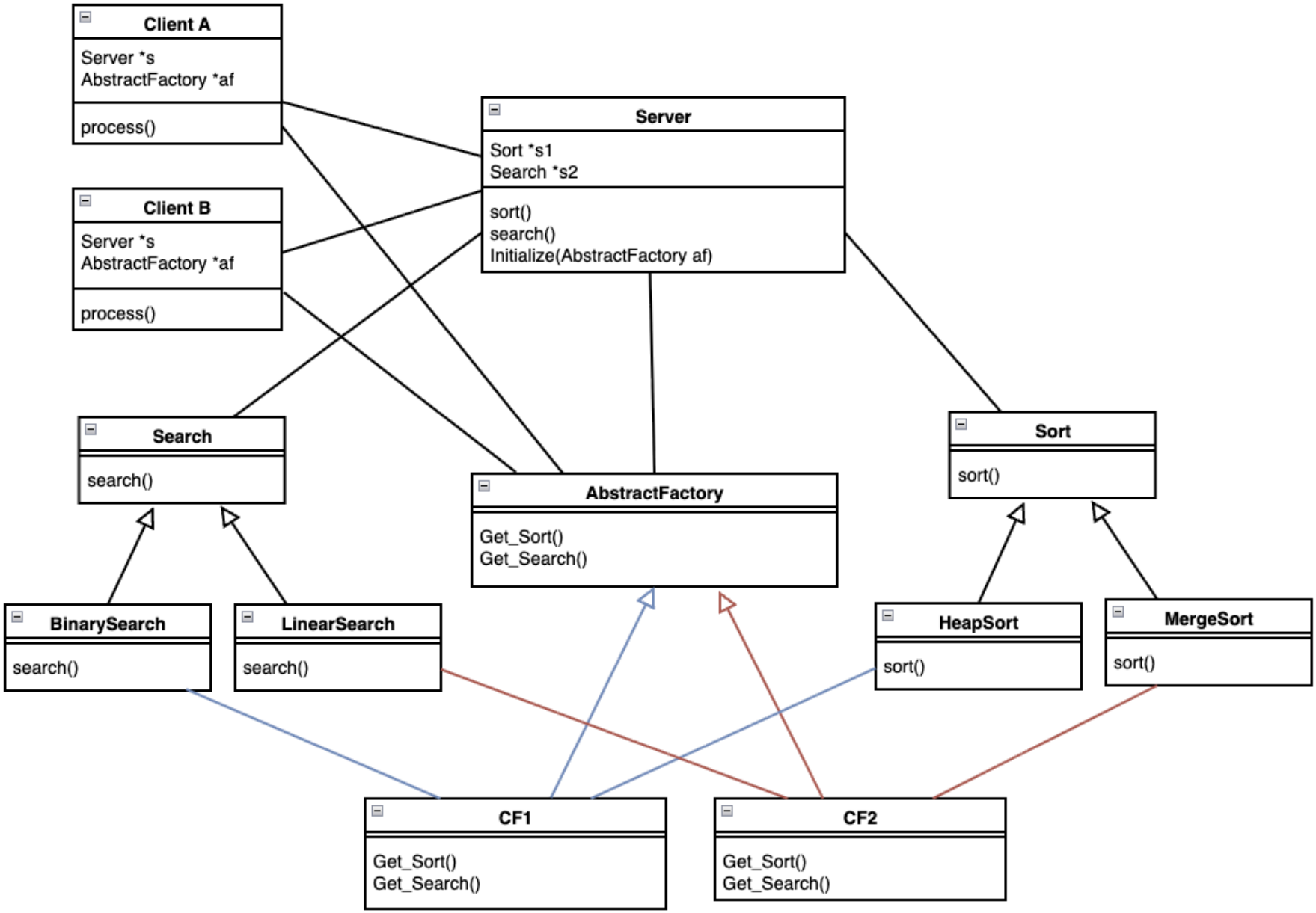
Problem #2
Inheritance_based



Problem #2: Inheritance version: Pseudocode

```
Class S1_S2_1_Adapter{
Service1(){Service2()} }

Class S1_S2_2_Adapter{
Service1(){Service2()} }

Class S2_S3_1_Adapter{
Service2(){Service3()} }

Class S2_S3_2_Adapter{
Service2(){Service3()} }

Class S3_S1_1_Adapter{
Service3(){Service1()} }

Class S3_S1_2_Adapter{
Service3(){Service1()} }
```

Problem #3


Client A
Server *s
AbstractFactory *af
process()
Client B
Server *s
AbstractFactory *af
process()
Server
Sort *s1
Search *s2
sort()
search()
Initialize(AbstractFactory af)
Search
search()
AbstractFactory
Get_Sort()
Get_Search()
Sort
sort()
BinarySearch
search()
LinearSearch
search()
HeapSort
sort()
MergeSort
sort()
CF1
Get_Sort()
Get_Search()
CF2
Get_Sort()
Get_Search()

Problem 3 Pseudocode

Class Client A{
Server *s
AbstractFactory *af

process(){
Af = new CF1()
s=> initialize(af)
s=> sort()
s=>search()
}
}

Class Client B{
Server *s
AbstractFactory *af

process(){
Af = new CF2()
s=> initialize(af)
s=> sort()
s=>search()
}
}

Class Server{
sort *s1
search *s2

initialize(AbstractFactory *af){
s1 = af=> Get_Sort()
s2 = af=> Get_Search()
}
Sort(){
s1=> sort()
}
Search(){
s2=> search()
}**}**

Class abstractFactory{ *// abstract class*
Get_Sort() and Get_Sort() are abstract methods.
}

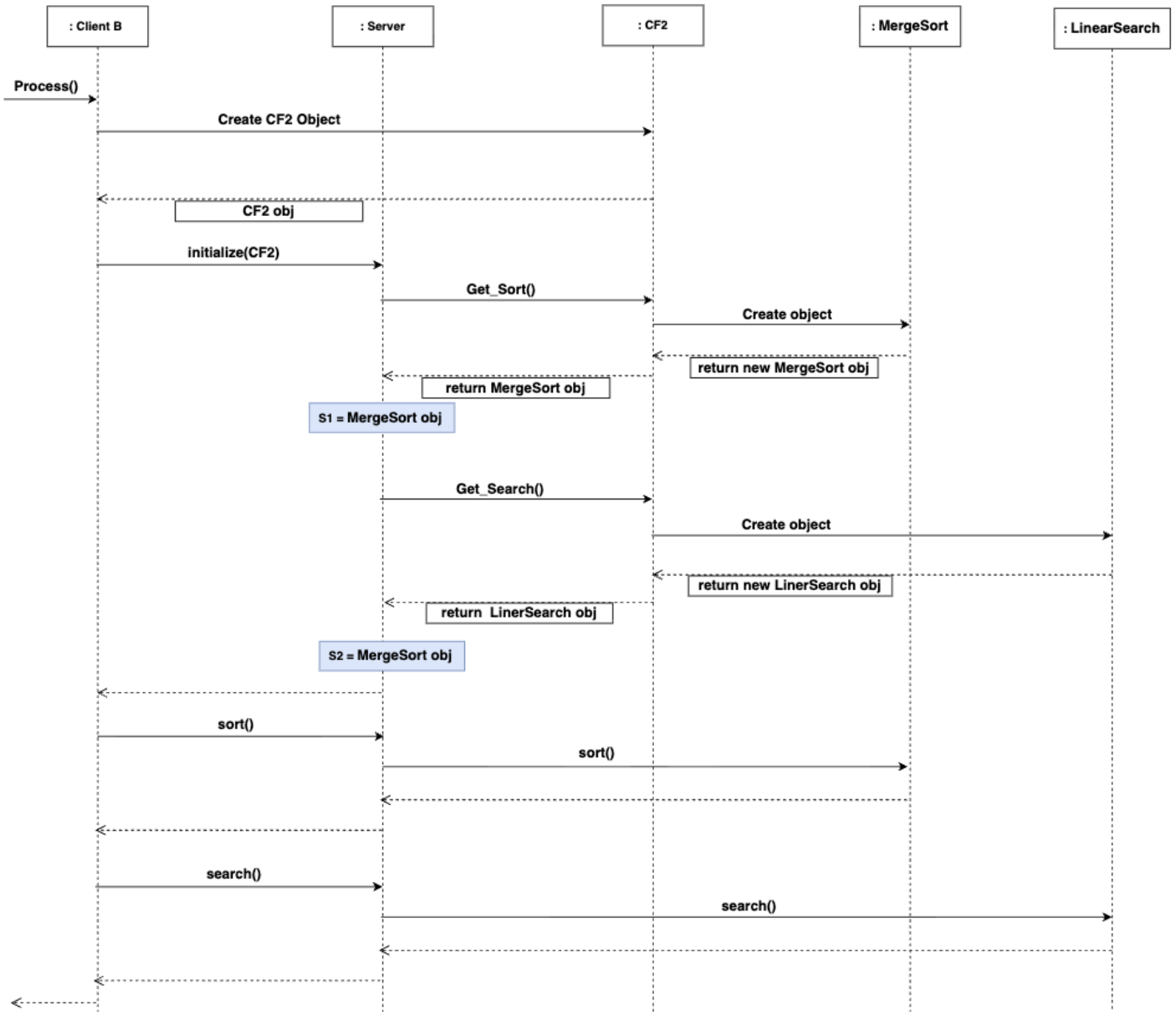
Class CF1{
Get_Sort(){
return new HeapSort()
}

Get_Search(){
return new BinarySearch()
}
}

Class CF2{
Get_Sort(){
return new MergeSort()
}

Get_Search(){
return new LinearSearch()
}
}

Problem 3 Sequence Diagram



End Of File….