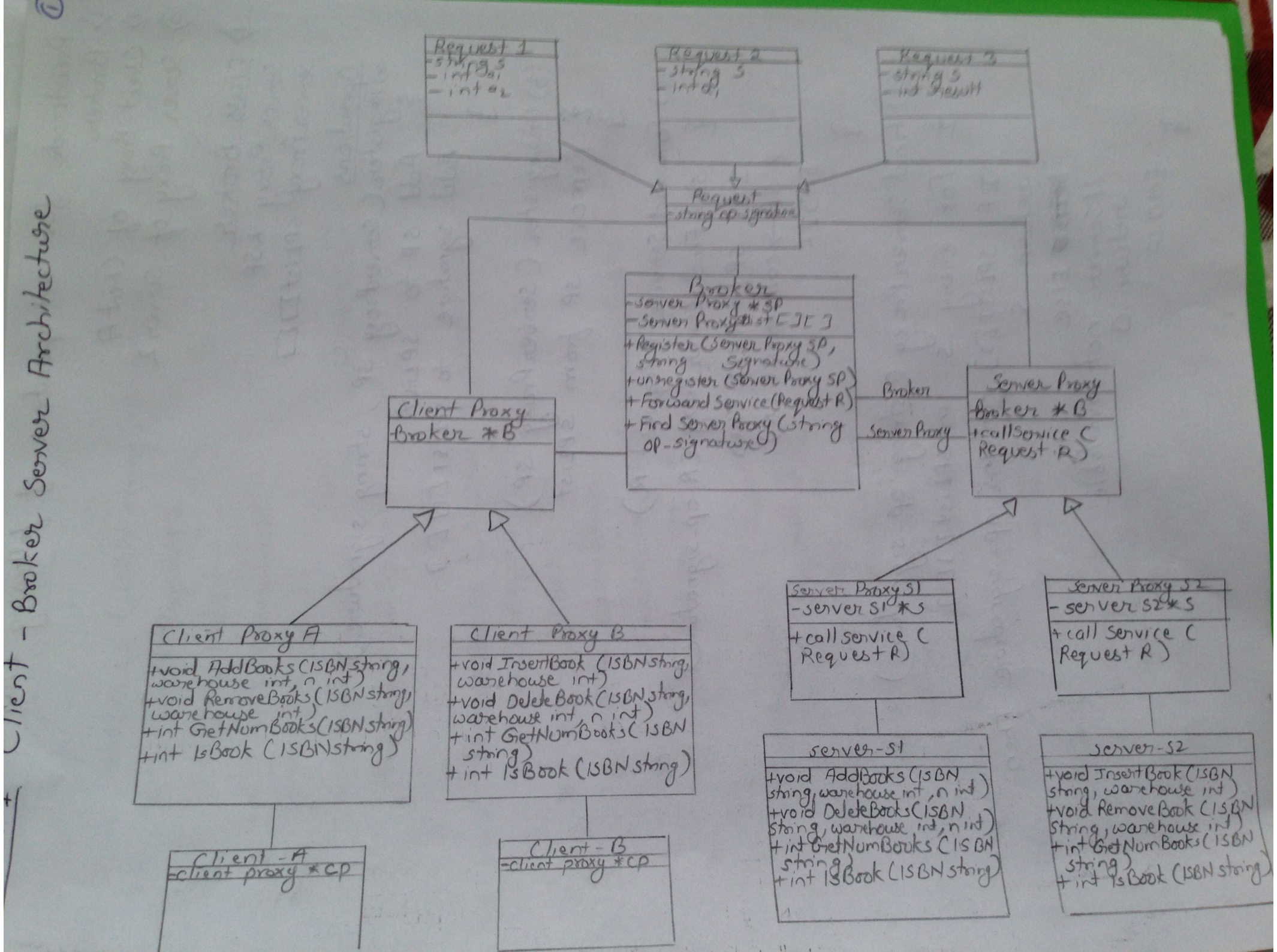

Client - Broker Server Architecture
Request 1
-string s
-int a1
-int a2
Request 2
-string s
-int b1
Request 3
-string s
-int c1
-int c2
Request
-string op-signature
Broker
-Server Proxy *SP
-Server Proxy list[3][3]
+Register(Server Proxy SP, string Signature)
+unregister(Server Proxy SP)
+Forward Service(Request R)
+Find Server Proxy(string op-signature)
Broker
Server Proxy
Server Proxy
Broker *B
+callService(Request R)
Client Proxy
Broker *B
Client Proxy A
+void AddBooks(ISBN string, warehouse int, n int)
+void RemoveBooks(ISBN string, warehouse int)
+int GetNumBooks(ISBN string)
+int IsBook(ISBN string)
Client Proxy B
+void InsertBook(ISBN string, warehouse int)
+void DeleteBook(ISBN string, warehouse int, n int)
+int GetNumBooks(ISBN string)
+int IsBook(ISBN string)
Server Proxy S1
-server S1 *S
+call service(Request R)
Server Proxy S2
-server S2 *S
+call service(Request R)
Client - A
-client proxy *CP
Client - B
-client proxy *CP
server-S1
+void AddBooks(ISBN string, warehouse int, n int)
+void DeleteBooks(ISBN string, warehouse int, n int)
+int GetNumBooks(ISBN string)
+int IsBook(ISBN string)
server-S2
+void InsertBook(ISBN string, warehouse int)
+void RemoveBook(ISBN string, warehouse int)
+int GetNumBooks(ISBN string)
+int IsBook(ISBN string)

Pseudocode for all operations of following classes

1) Broker
2) Client Proxy of Client A
3) Server Proxy of Server 2

1) Class Broker

ServerProxy *SP
ServerProxy SPList[][]

Operations

(a) Register (ServerProxy SP, string signature)
{ Add SP to SPList
Add signature to SPList[SP][ ]
}

(b) Unregister (ServerProxy SP)
{ Remove SP from SPList
}

(c) Forward Service (Request R)
{ SP = Find ServerProxy (R.op-signature)
IF SP != NULL Then
SP → callservice (R)
END IF
}

(d) Find ServerProxy (string op-signature)
{ For every s in SPList[][]
IF SPLIST[s] contains op-signature Then
return s
Else
// server not available
return 0
END IF
}

Client Proxy A class operations

(a) void AddBooks (ISBN string, warehouse int, n int)
{ R = new Request1
R→s = ISBN
R→a_1 = warehouse
R→a_2 = n
R→op_signature = "void AddBooks (string, int, int)"
B→ForwardService (R)
}

(b) void RemoveBooks (ISBN string, warehouse int)
{ R = new Request2
R→s = ISBN
R→a_1 = warehouse
R→op-signature = "void RemoveBooks (string, int)"
B→ForwardService (R)
}

(c) int GetNumBooks (ISBN string)
{ R = new Request3
R→s = ISBN
R→op-signature = "int GetNumBooks (string)"
B→ForwardService (R)
return R→result
}

(d) int IsBook (ISBN string)
{ R = new Request3
R→s = ISBN
R→op-signature = "int IsBook (string)"
B→ForwardService (R)
return R→result

③ class ServerProxyS2
server S2 *S

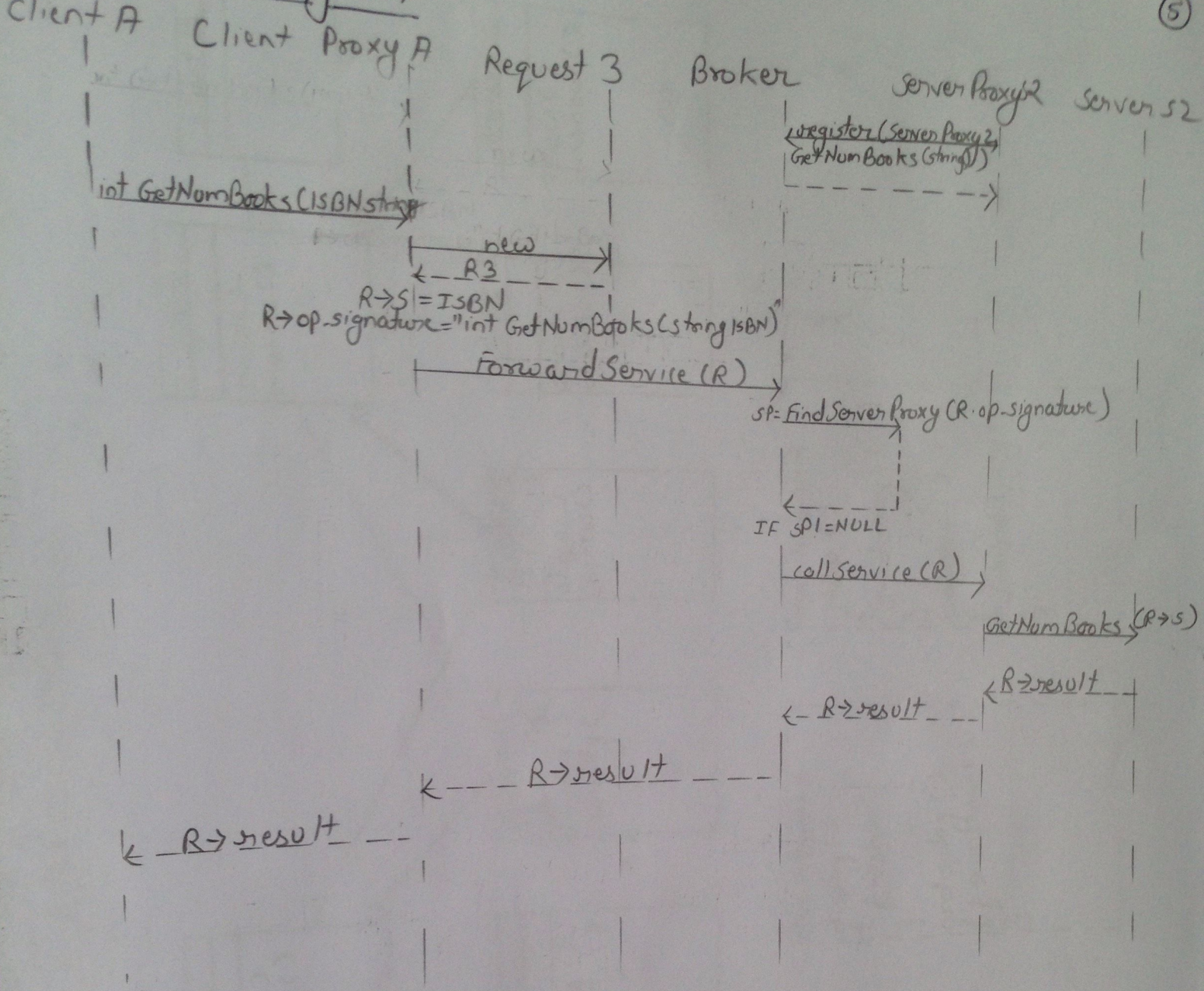
Operations

(a) call service (Request R)
{ IF (R→op-signature == "void InsertBook (string, int)")
Then S→InsertBook (R→s, R→a,)
Else
IF (R→op-signature == "void RemoveBook (string, int)") then
S→RemoveBook (R→s, R→a,)
Else
IF (R→op-signature == "int GetNumBooks (string)" then
R→result = S→GetNumBooks (R→s)
Else
IF (R→op-signature == "int IsBook (string)" then
R→result = S→IsBook (R→s)

End IF

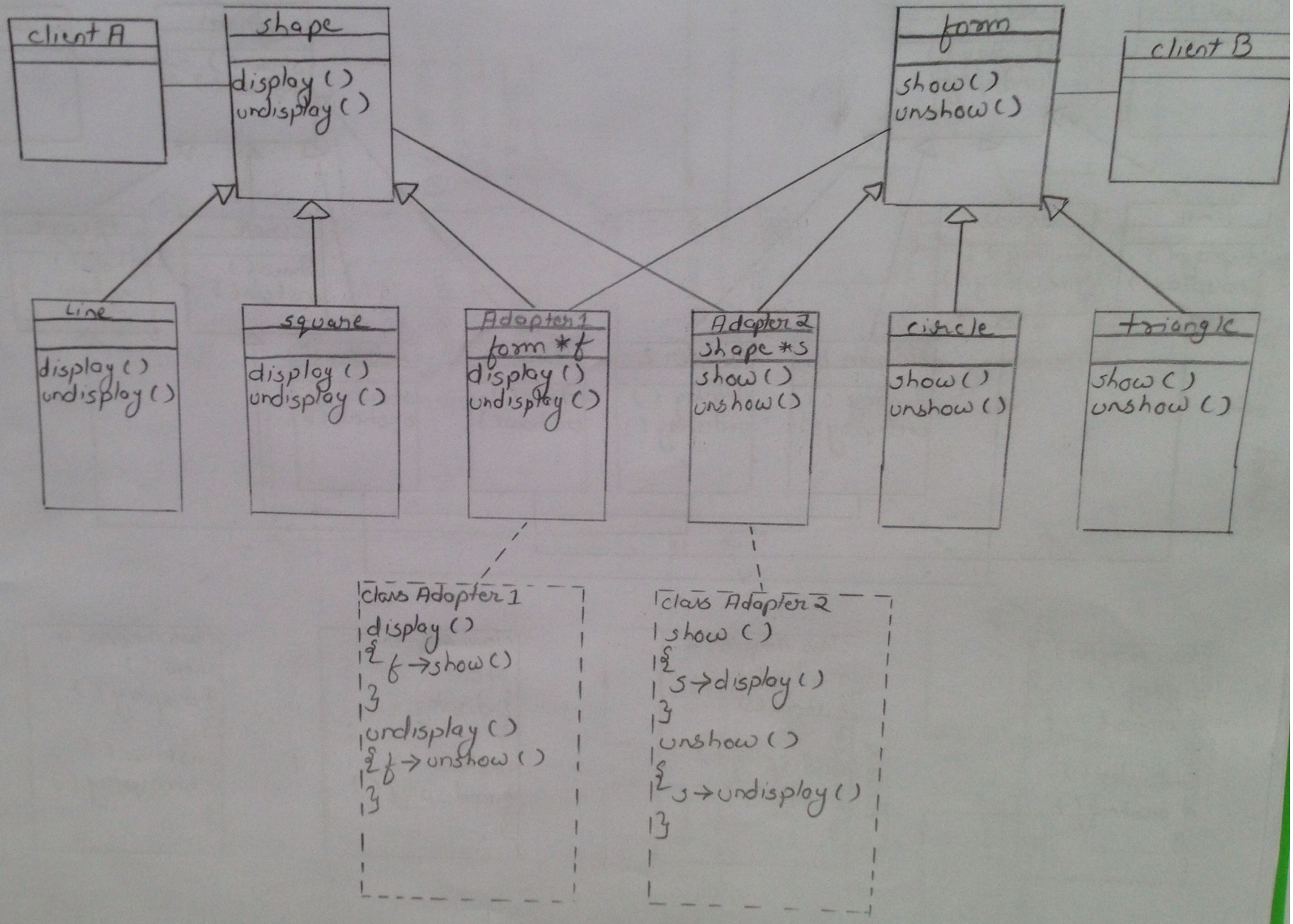
}

Sequence Diagram

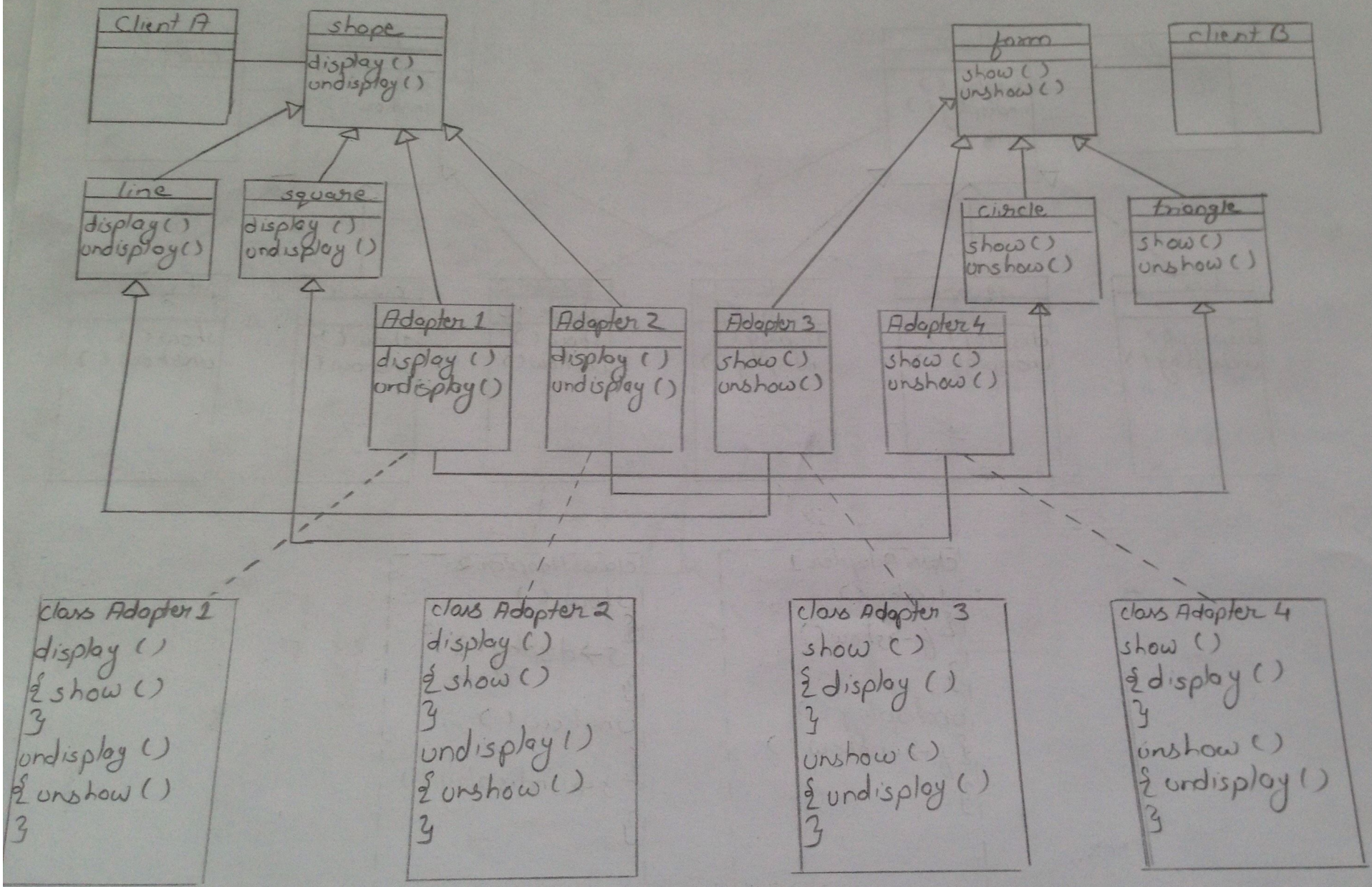


Problem-2 Adopter Pattern

Association Based



Inheritance Based



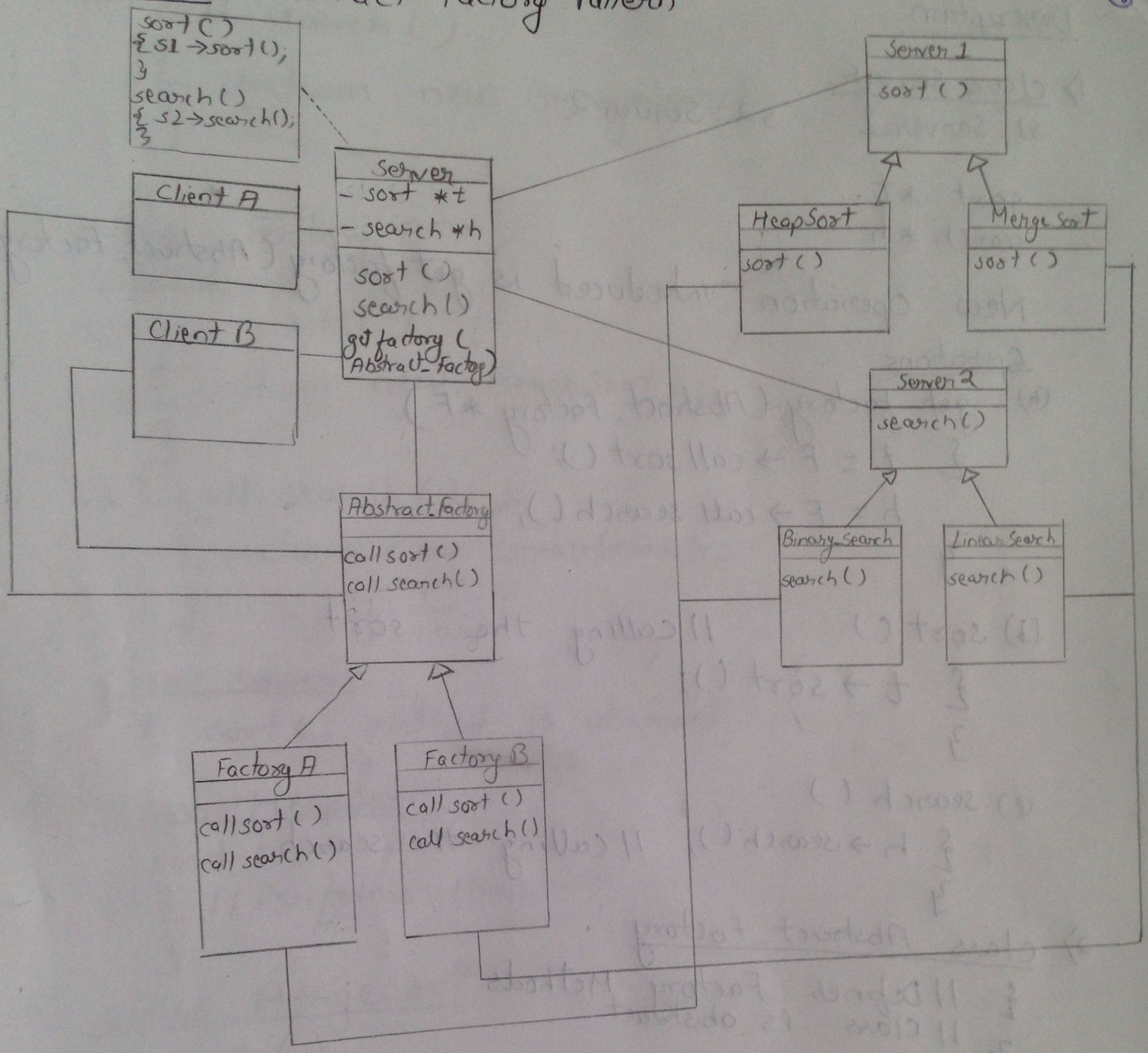

```
class Adapter 1
display()
{ show()
}
undisplay()
{ unshow()
}
```

```
class Adapter 2
display()
{ show()
}
undisplay()
{ unshow()
}
```

```
class Adapter 3
show()
{ display()
}
unshow()
{ undisplay()
}
```

```
class Adapter 4
show()
{ display()
}
unshow()
{ undisplay()
}
```

Problem-3 Abstract Factory Pattern



1) clas Server.

s1 : Server 1 s2 : Server 2

sort *t
search *h

New Operation introduced is get factory (Abstract_Factory)

Operations

(a) get factory (Abstract_Factory *F)
{ t = F → call sort ();
h = F → call search ();
}

(b) sort () // calling the sort
{ t → sort ();
}

(c) search ()
{ h → search (); // calling the search
}

2) clas Abstract_Factory
{ // Defines Factory Methods
// clas is abstract
}

3) clas Factory A.

Operations

(a) call sort ()
{ return new HeapSort ();
//new operator is used to encapsulates the Factory methods
}

(b) call search()
{ return new BinarySearch;
}

④ clas Factory B
Operations
(a) call sort()
{ return new MergeSort;
}
(b) call search()
{ return new LinearSearch;
}

⑤ clas Server1
sort() Method is abstract

⑥ clas HeapSort
sort()
{ // Performs HeapSort
}

⑦ clas MergeSort
sort()
{ // Performs MergeSort
}

⑧ clas Server2
Search() Method is abstract

```
class BinarySearch
search()
{ //Performs Binary Search
}
```

(10)
```
class LinearSearch
search()
{ //Performs Linear Search
}.
```

Sequence Diagram