Software Systems Architecture (CS-586)

Homework #3

- *Sivasenthil Namachivayan*

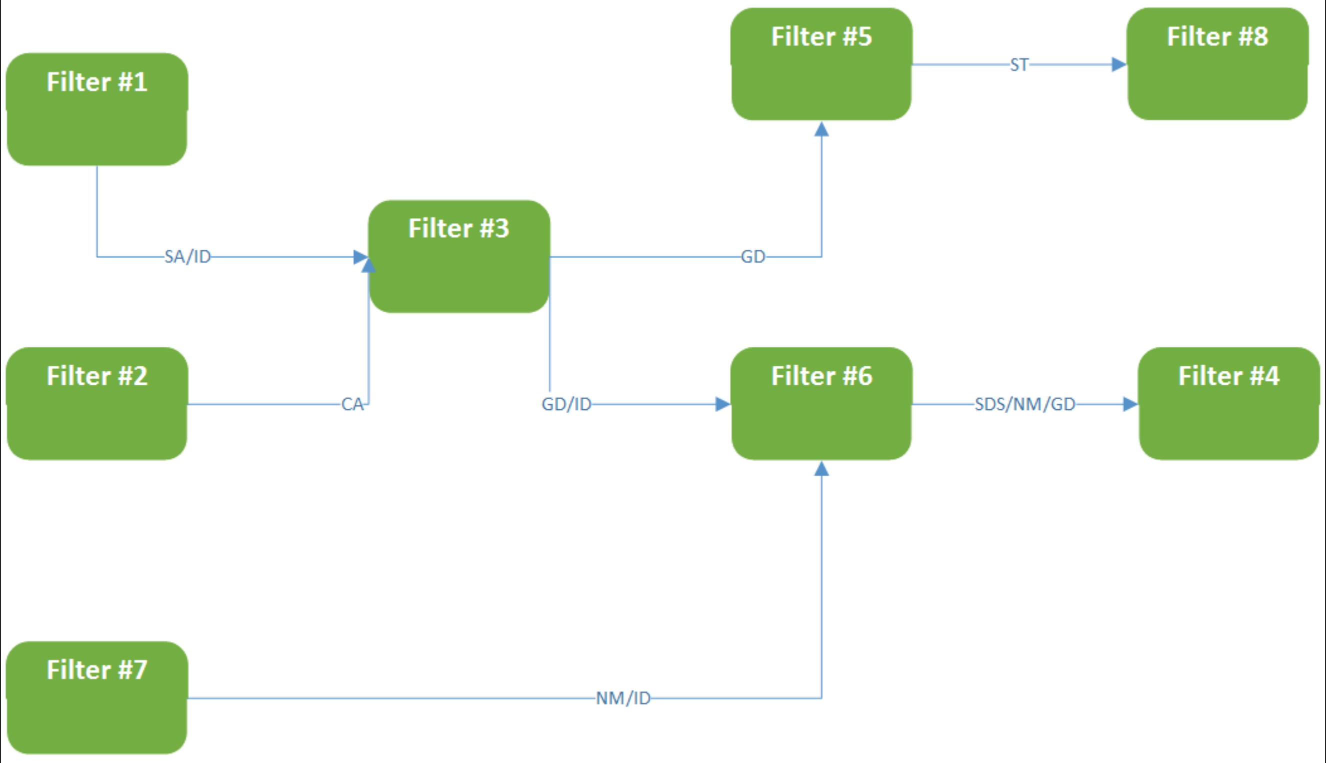
- *A20391478*

Table of Contents

Problem #1

Pipes and Filters

Part A:



Filters

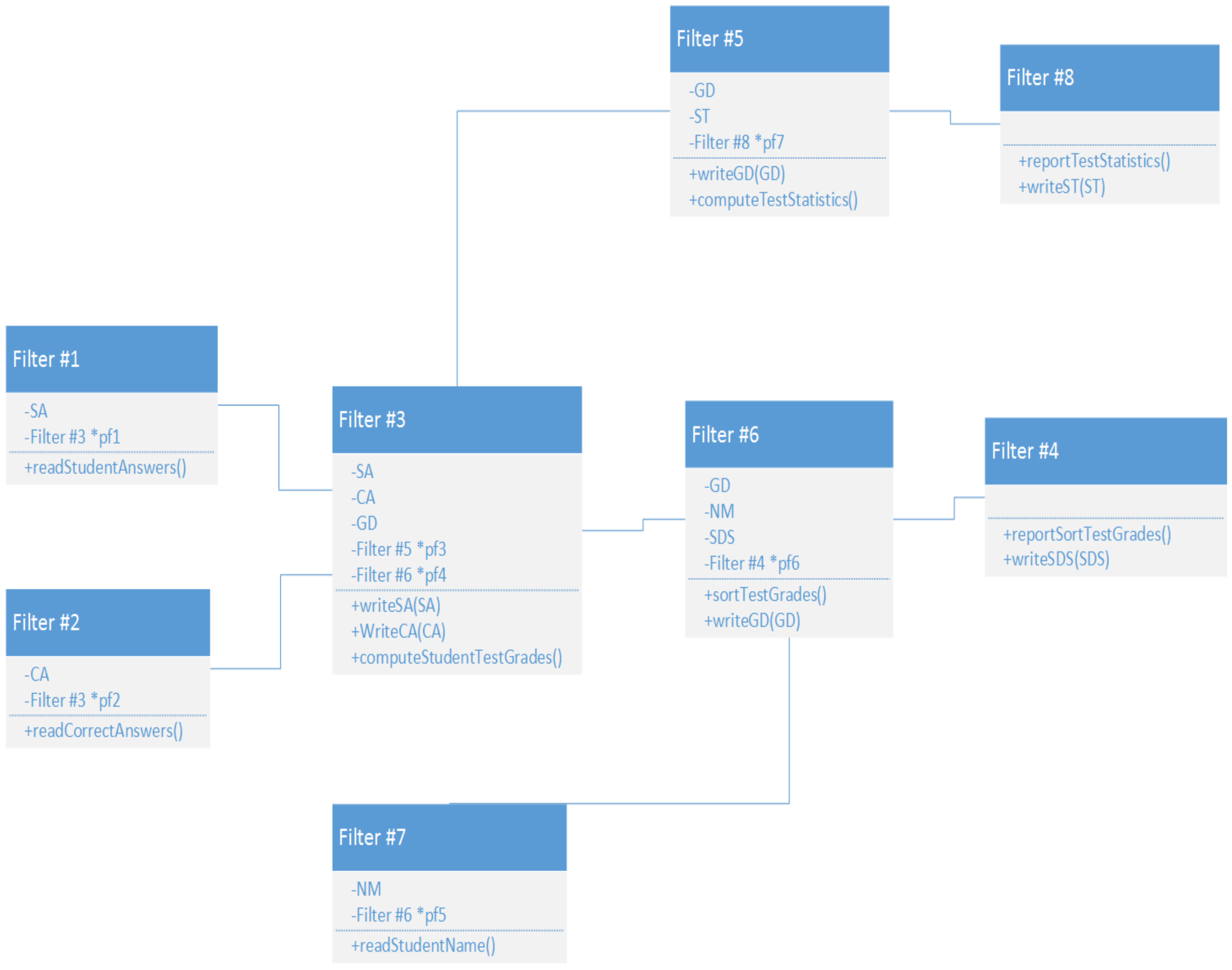
Filter #1: this filter reads student's test answers together with student's IDs
Filter #2: this filter reads correct answers for the test
Filter #3: this filter computes test grades with student ID
Filter #4: this filter prints test grades with student names in the order as they are read from an input pipe
Filter #5: this filter computes test statistics
Filter #6: this filter sort student names in the descending order with respect to the grades
Filter #7: this filter reads the student's name together with their IDs
Filter #8: this filter reports the test statistics

Pipes

SA: Student's test answers together with student's ID
CA: Correct answers for the test
GD: Student's test grades
NM: Student's name together with student's ID
ST: Test Statistics (# of A Grades, # of b Grades)
SDS: Student names sorted in descending order with respect to the grades

Part B:

2. Class Diagram



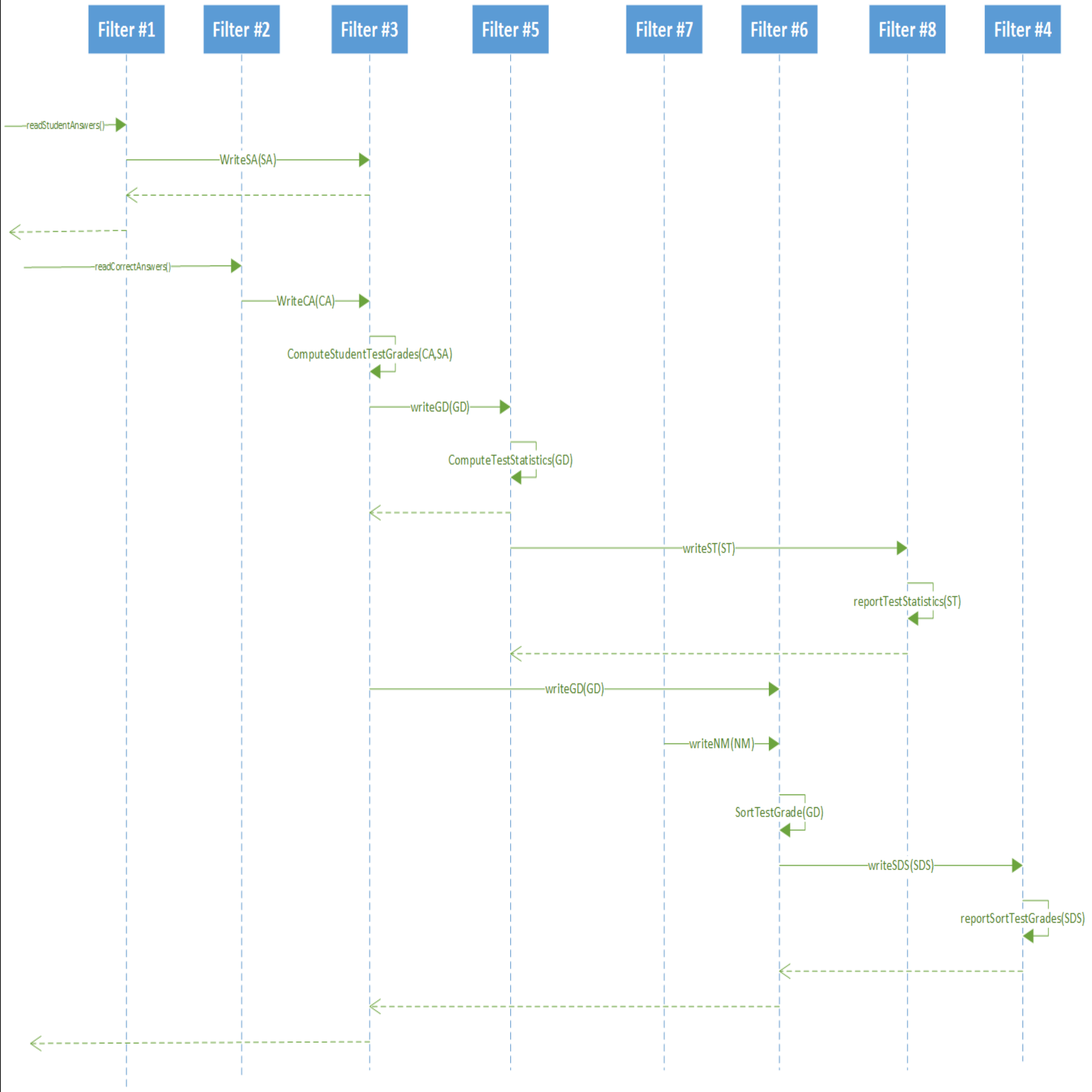
Pseudo-code

```
Class Filter #1
Filter #3 *pf1
SA- Student's test answers together with the student's ID
readStudentAnswers()
{
        read students test answers together with the student ID into SA;
        pf1 → writeSA(SA);
}

Class Filter #2
Filter #3 *pf2
CA- Correct answers for the test
readCorrectAnswers()
{
        read correct answers into CA;
        pf2→ writeCA(CA);
}

Class Filter #3
SA- Student’s test answers along with student’s ID
CA- Correct Answers for the test
GD- Test Grades(A,B,C,E) along with the student’s ID
Filter #5 *pf3
Filter #6 *pf4
flagCA=false;
flagSA=false;
writeSA(SA)
{
        1. Store into SA
        2. check if (flagCA==true)
        //CA is in
        call computeStudentGrades()
                flagCA=false;
                flagSA=false;
        else flagSA=true;
}
writeCA(CA)
{
        1. Store into CA
        2. check if (flagSA==true)
        //SA is in
        call computeStudentGrades()
                flagCA=false;
                flagSA=false;
        else flagCA=true;
}
ComputeStudentGrades()
{
//flags will be true when we have both the data structures SA and CA
        GD= computeStudentTestGrades(CA,SA)
        pf3→ writeGD(GD)
        pf4→ writeGD(GD)
}
```

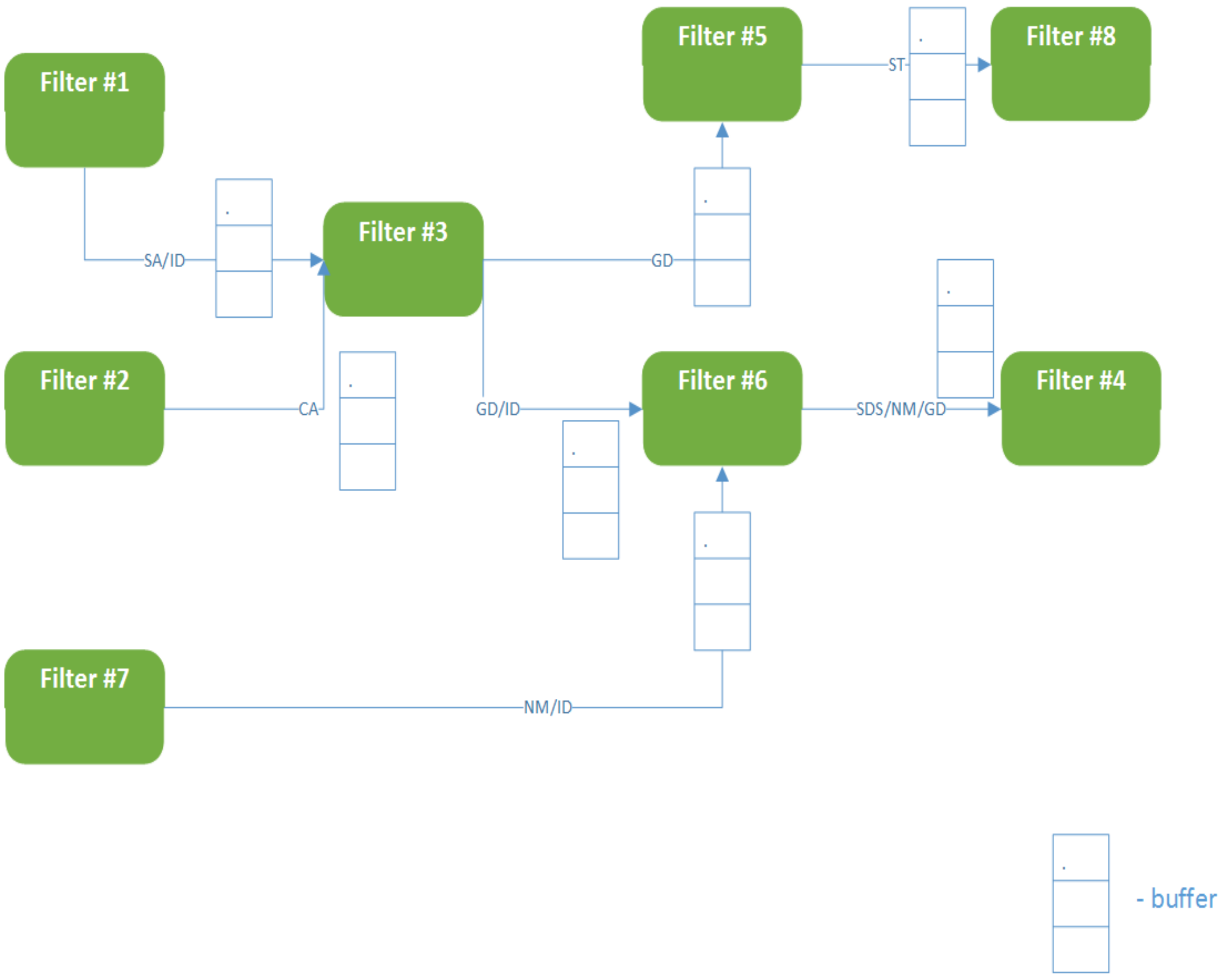
```
computeStudentTestGrades(CA,SA)
{
        grade Student Answers into GD
}

Class Filter #4
SDS- Students name sorted in descending order with respect to Grades received
writeSDS(SDS)
{
        1. Store into SDS
        2. //SDS is in
        call reportSortTestgrades()
}

reportSortTestGrades()
{
        print student names sorted in descending order with respect to the grades (descending order of
                name with grade A, descending order of Name with Grade B and descending order of
                name with grade C)
}

Class Filter #5
Filter #8 *pf7
GD- Student Grade
ST- Student Test Statistics (# of A grades, # of B Grades)
WriteGD(GD)
{
1. Store into GD
2. //GD is in
        call ComputeTestStatistics()
}

computeTestStatistics()
{
        //flag will be true when we have the data structure GD compute statistics into ST using GD
        compute and return test statistics (Number of students who got A grade, Number of students
                got B, Number of students got C and number of students got E)
        pf7→ writeST(ST)
}

Class Filter #6
GD- Student Grade
NM- Student Name received from Filter #7
SDS- Students names sorted in descending order with respect to the grades received
Filter #4 *pf6
WriteGD(GD)
```

```
{
1. Store into GD
2. //GD is in
        call SorttestGrades()
}

sortTestGrades()
{
        Sort student names in descending order of the name with respect to the grades
        pf6→ writeSDS(SDS)
}

Class Filter #7
NM- Student Names
Filter #6 *pf5
readStudentName()
{
        read Student names into NM;
        pf5→ writeNM(NM);
}

Class Filter #8
ST- Students grade statistics (# of students got A, # of students got B, # of students got C and # of
students got E)
WriteST(ST)
{
1. Store into ST
2. //ST is in
        call reportTestStatistics()
}
reportTestStatistics()
{
        report/print the statistics
}
```

3. Sequence Diagram



Part C:

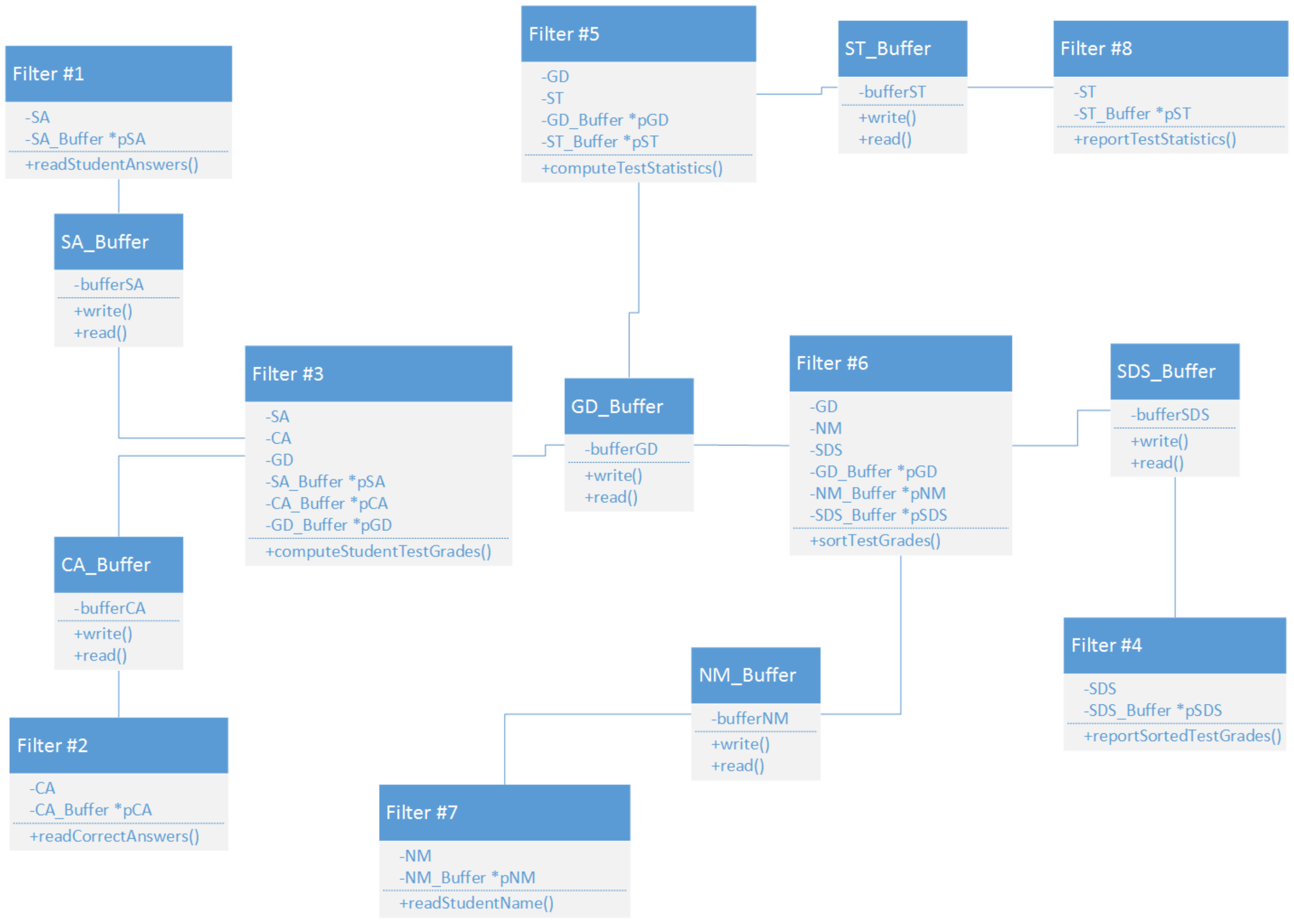


Filters

Filter #1: this filter reads student's test answers together with student's IDs
Filter #2: this filter reads correct answers for the test
Filter #3: this filter computes test grades with student ID
Filter #4: this filter prints test grades with student names in the order as they are read from an input pipe
Filter #5: this filter computes test statistics
Filter #6: this filter sort student names in the descending order with respect to the grades
Filter #7: this filter reads the student's name together with their IDs
Filter #8: this filter reports the test statistics

Pipes
SA: Student’s test answers together with student’s ID
CA: Correct answers for the test
GD: Student’s test grades
NM: Student’s name together with student’s ID
ST: Test Statistics (# of A Grades, # of B Grades)
SDS: Student names sorted in descending order with respect to the grades

2. Class Diagram



Pseudo-code

```
Class Filter #1
SA- Student's test answers together with the student's ID
SA_Buffer *pSA
readStudentAnswers()
{
Loop
    read students test answers together with the student ID into SA;
    pSA→write(SA);
EndLoop
}

Class Filter #2
CA- Correct answers for the test
CA_Buffer *pCA
readCorrectAnswers()
{
Loop
    read correct answers into CA;
    pCA→write(CA); // Correct answers for the test
EndLoop
}

Class Filter #3
SA- Student's test answers along with student's ID
CA- Correct Answers for the test
GD- Test Grades(A,B,C,E) along with the student's ID
SA_Buffer *pSA
CA_Buffer *pCA
GD_Buffer *pGD

computeStudentTestGrades()
{
Loop
    SA = pSA→ read() // read Student's test answers together with the student ID
    CA= pCA→ read() // read Correct answers
    computes test grades with SA and CA
    put student's test grade together with students ID into GD
    pGD→ write(GD);
EndLoop
}

Class Filter #4
SDS- Students name sorted in descending order with respect to Grades received
SDS_Buffer *pSDS
reportSortedTestGrades()
```

```
{
Loop
        SDS= pSDS→ read()// read student names sorted in descending order with respect to the
                                    grades
        print SDS
EndLoop
}

Class Filter #5
GD- Student Grade
ST- Student Test Statistics (# of A grades, # of B Grades)
GD_Buffer *pGD
ST_Buffer *pST

computeTestStatistics()
{
Loop
        GD= pGD→ read() // read student's test grade from pipe GD
        compute test statistics with GD
        return ST
EndLoop
}

Class Filter #6
GD- Student Grade
NM- Student Name received from Filter #7
SDS- Students names sorted in descending order with respect to the grades received
GD_Buffer *pGD
NM_Buffer *pNM
SDS_Buffer *pSDS
sortTestGrades()
{
Loop
        GD= pGD→ read() //read student's test grade with student ID
        Sort student names in descending order of the name with respect to the grades
        pSDS→ write(SDS)
EndLoop
}

Class Filter #7
NM- Student Names
NM_Buffer *pNM
readStudentName()
{
Loop
        read Student names into NM;
```

```
    pNM→ write(NM) // students Name
EndLoop
}
```

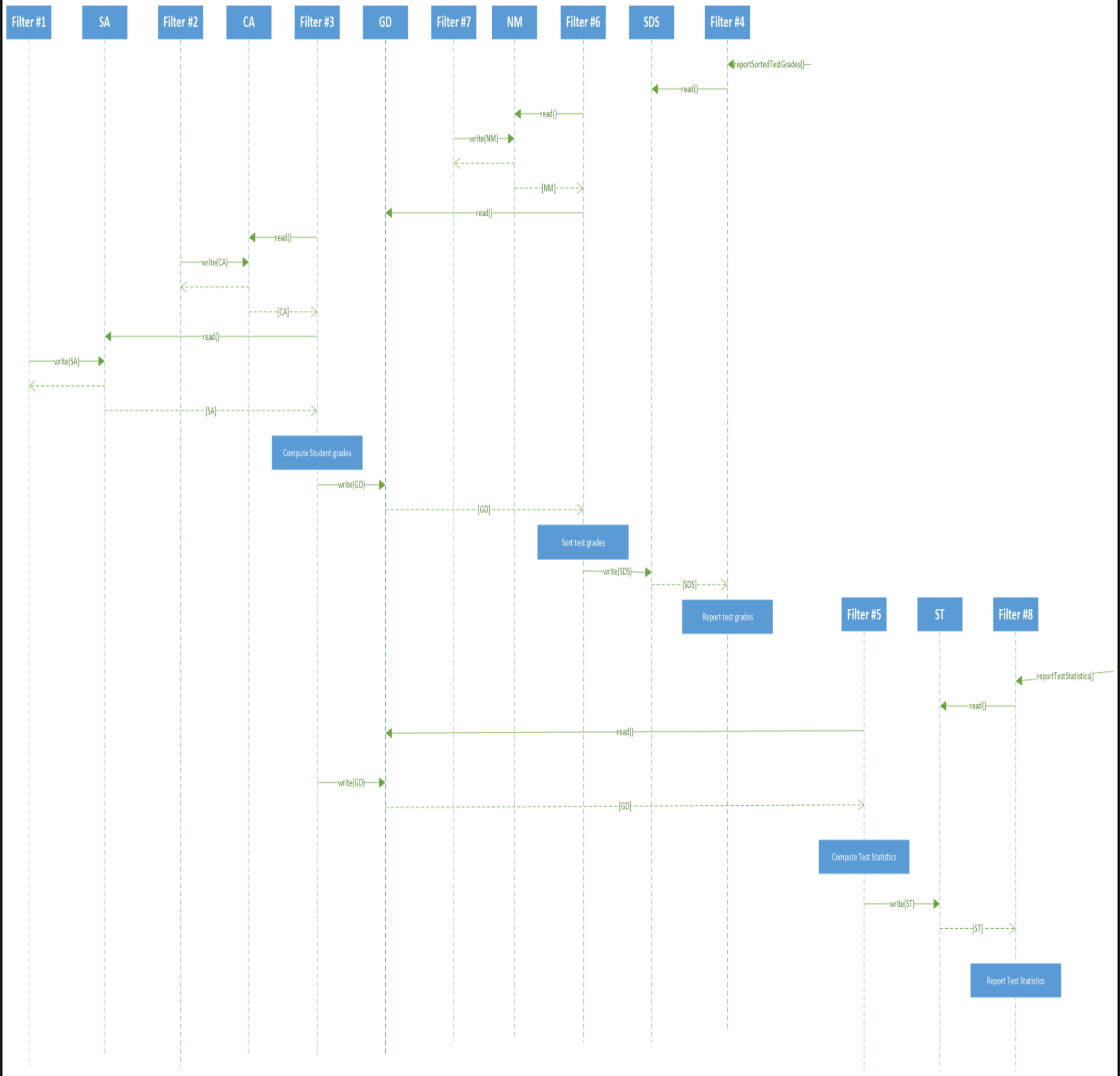
Class Filter #8
ST- Students grade statistics (# of students got A, # of students got B, # of students got C and # of students got E)

```
ST_Buffer *pST
reportTestStatistics()
{
Loop
    ST= pST→ read() read statistics
    print ST
EndLoop
}
```

Pipe Classes

```
buffer B
list read()
{
    if buffer B is empty then
        wait until the buffer is not empty
        delete list from buffer B
        return list
    EndIf
}

write(list)
{
    put list into buffer B
}
```

3. Sequence Diagram

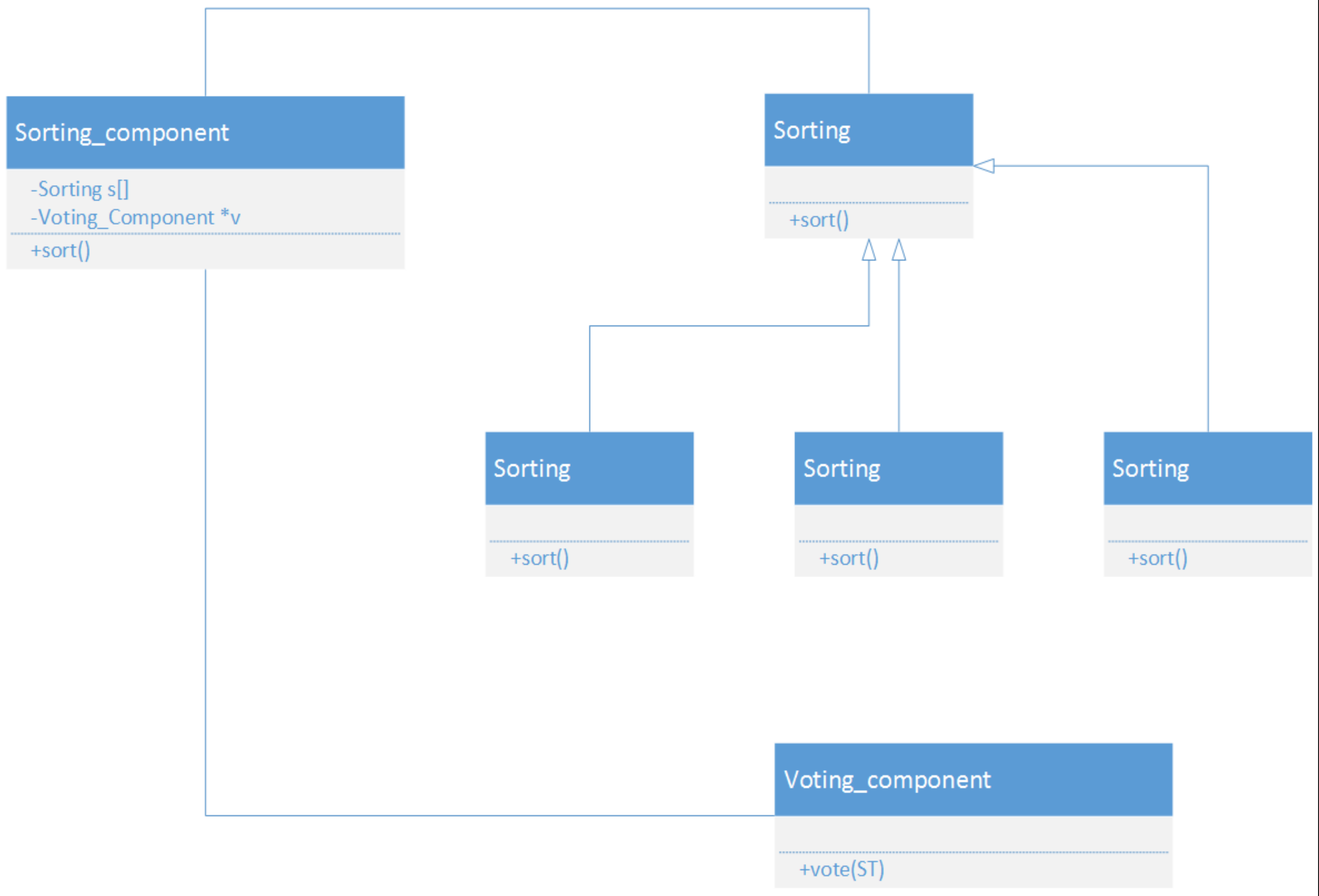


Problem #2

Fault Tolerant Architecture

1. N-Version Architecture

Class Diagram

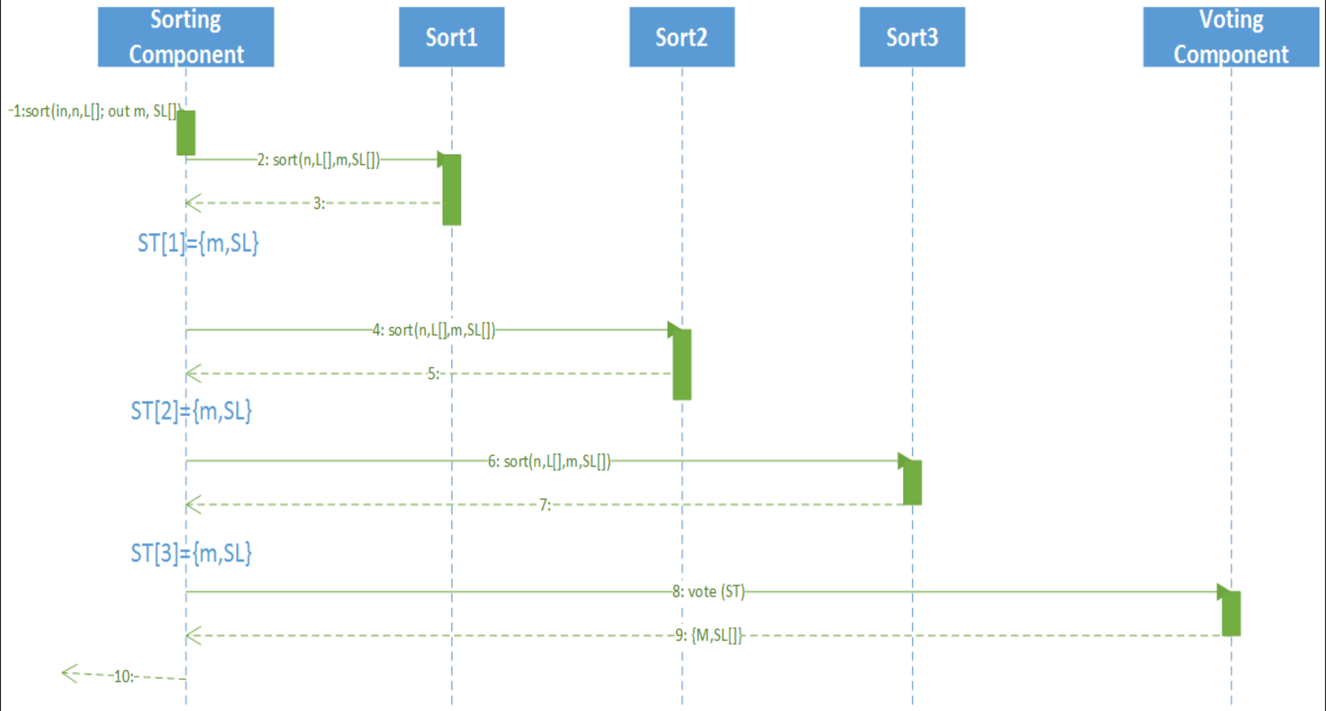


Pseudo-code

```
Class Sorting_Component
Sorting s[]
Voting_component *v
Void sort(in int n, int L[], out int m,int SL[])
{
ST[int,int]
```

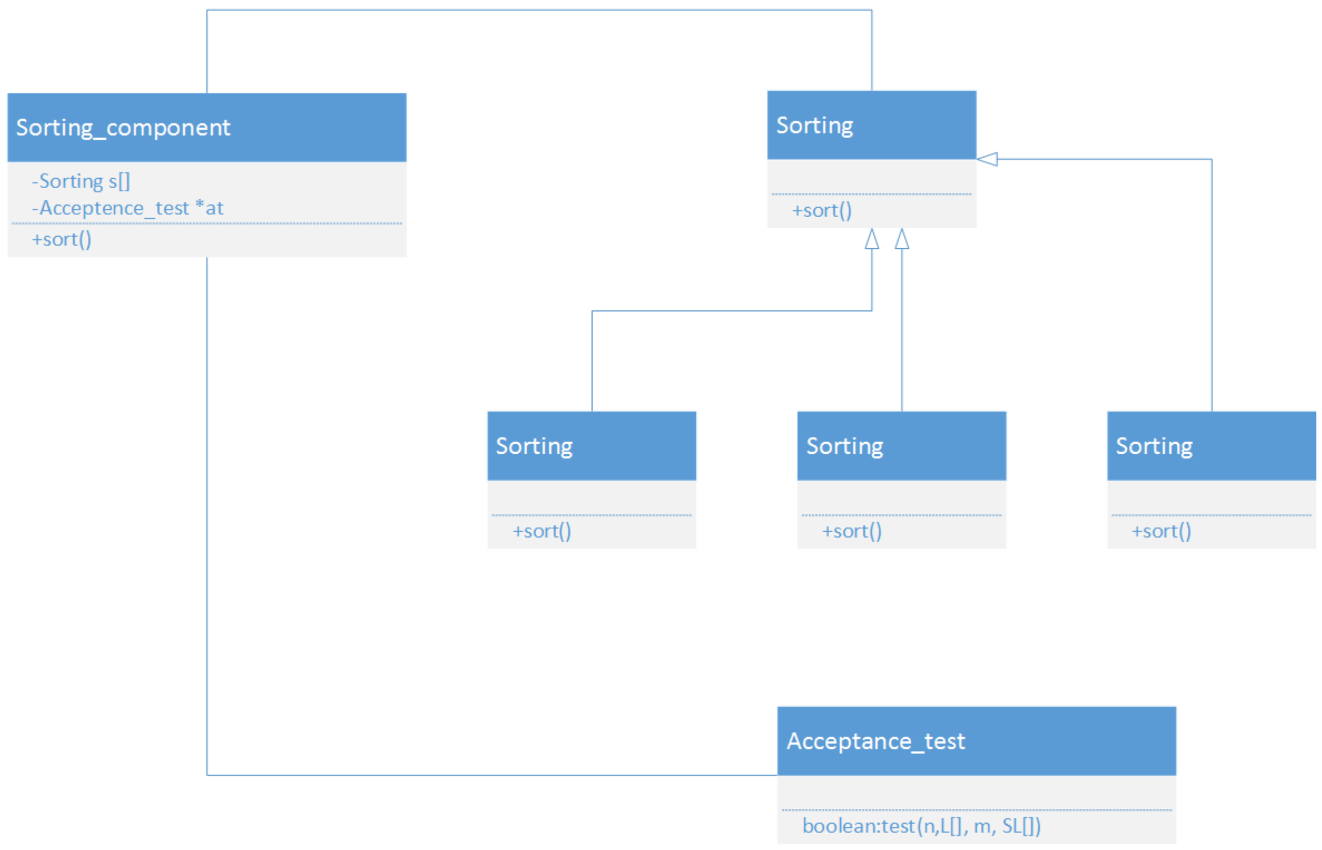
```
sorting s[];
s[1] = new sort1()
s[2]= new sort2()
s[3]= new sort3()
for Y=1 to 3
s[i]→ sort(n,L[],m,SL[])
ST[Y] = {m, SL[]}
End for
{m,SL[]}= v→ sort(ST)
}
Class Voting_Component:
{int,int} sort (in:SRT)
{
If ST[1] == ST[2] then
return ST[1]
else If ST[2]== ST[3] then
return ST[2]
else If ST[1] == ST[3] then
return ST[3]
endif
ran = random (1,3)
return ST[ran]
```

Sequence Diagram



2. Recovery- Block Architecture

Class Diagram

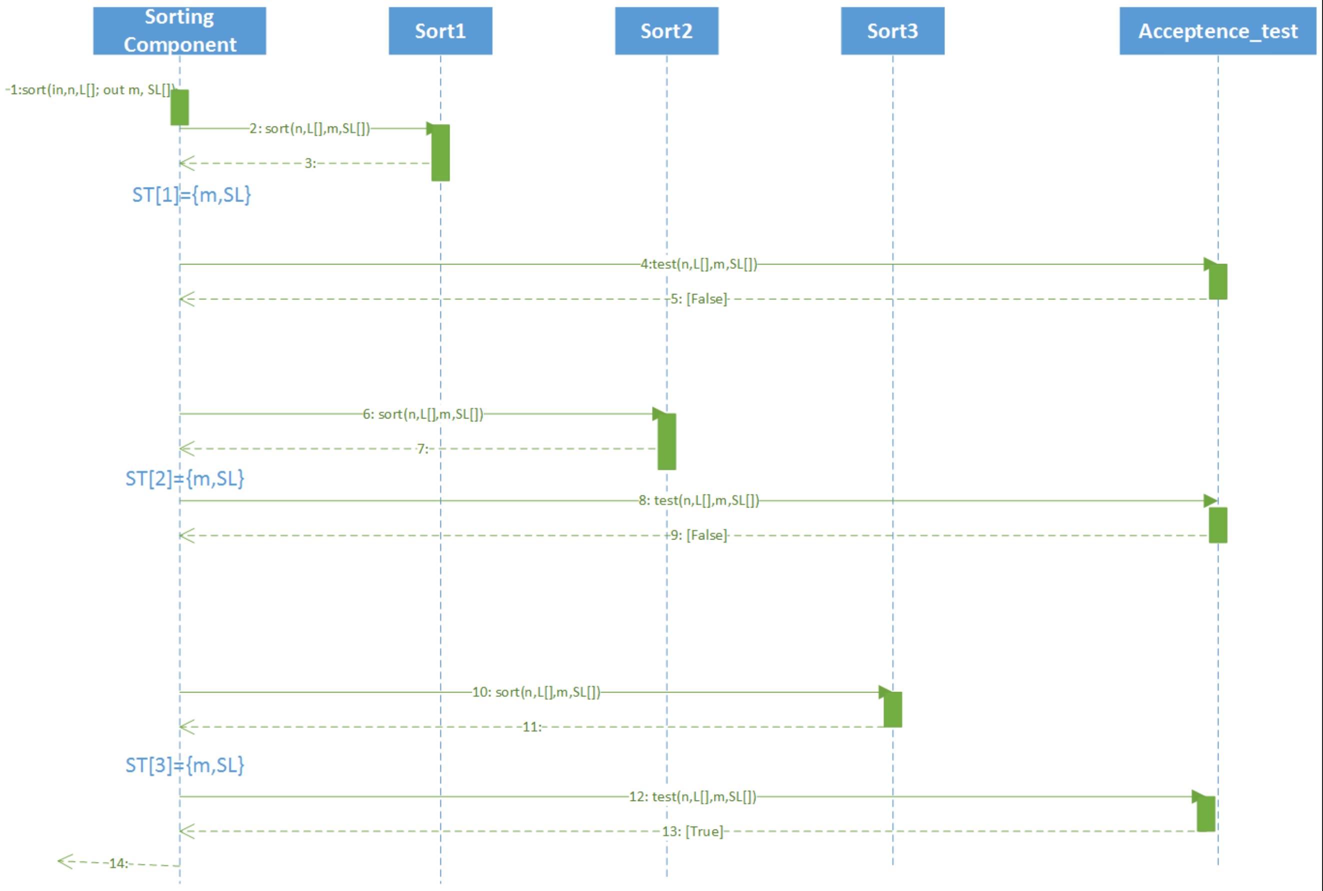


Pseudocode

```
Class acceptence_test
boolean test (in : int n, int L, out: int m, int SL)
{
For i=1 to n where n=number of outputs
if SL[i] > SL[i+1]
elseif SL[i+1]<0
then
return false
```

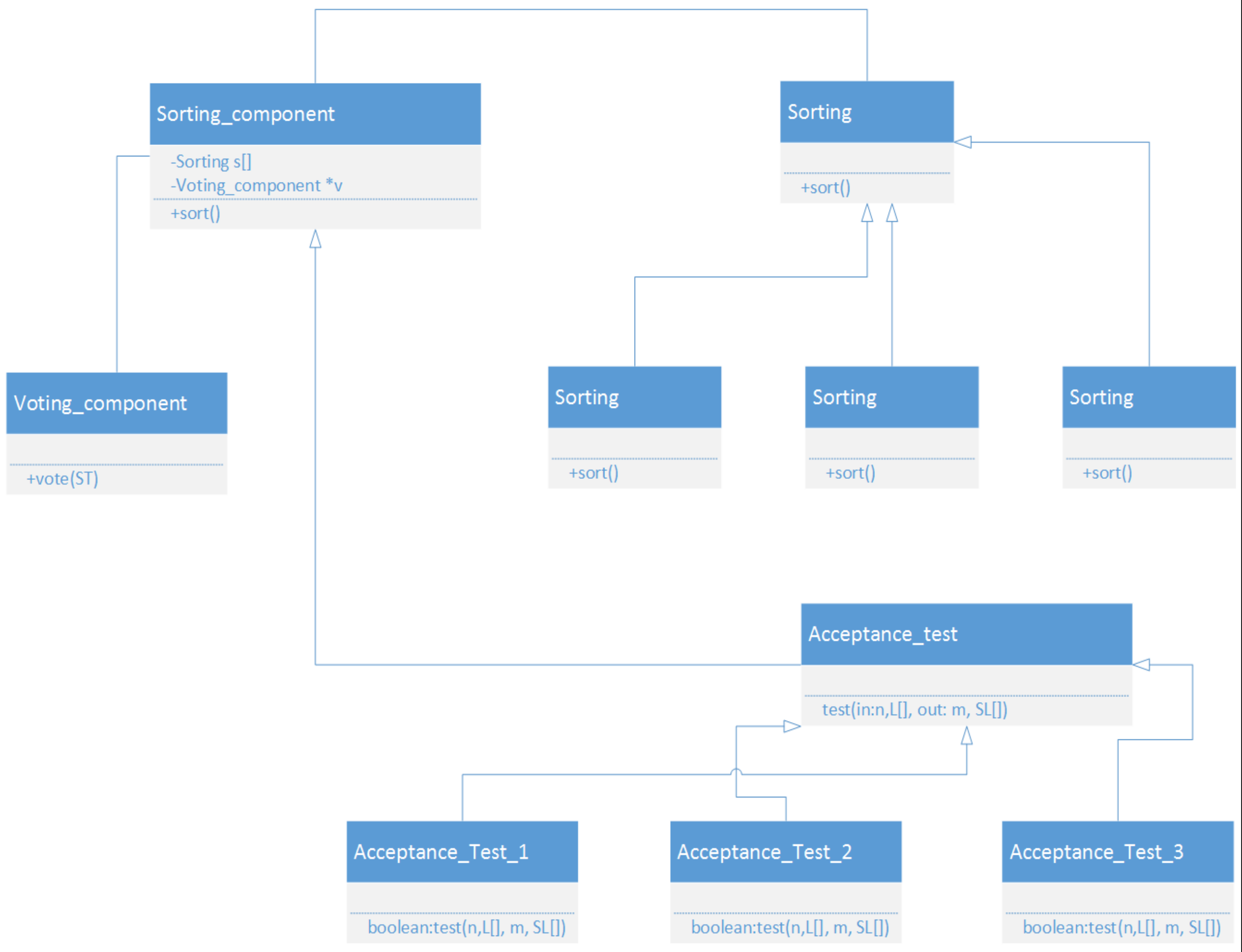
```
endif
endfor
Class Sorting_component
sorting s[]
acceptence_test *at
void sorting (in: int n, int L, int m,int SL)
{
{m,SL}ST[]
S[1] = new sort()
S[1]→ sort(n,L,m,SL)
ST[1] = {m.SL}
testsort = at→ test(n,L,m,SL)
if testsort == true then
exit
S[2] = new sort()
S[2]→ sort(n,L,m,SL)
ST[1] = {m.SL}
testsort = at→ test(n,L,m,SL)
if testsort == true then
exit
S[3] = new sort()
S[3]→ sort(n,L,m,SL)
ST[1] = {m.SL}
testsort = at→ test(n,L,m,SL)
if testsort == true then
exit
endif
ran = random(1,3)
{n,L}=ST[r]
}
```

Sequence Diagram



3. N-Self Checking Architecture

Class Diagram



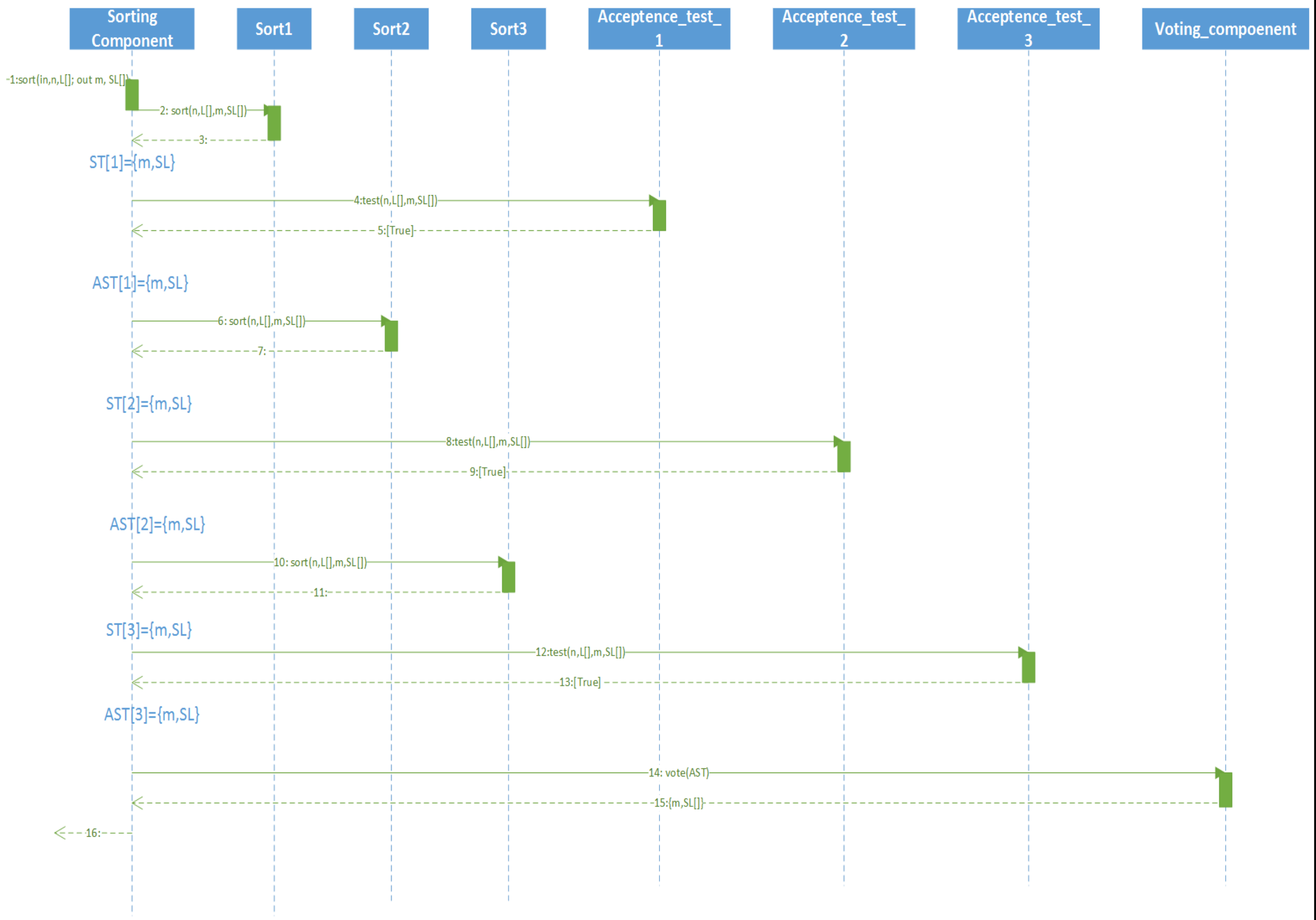
Pseudocode

```
Class Sorting_component
Sorting s[]
Acceptance_test *at
Voting_Component *v
void sort(in int n, int L[]; out int m, int SL[])
{
ST[] // an array of {int,int}
AST[] // an array of {int, int}
sorting s[]
Acceptance_test AT[]
S[1] = new sort(1)
AT[1] = new test(1)
S[2] = new sort(2)
AT[2] = new test(2)
S[3] = new sort(3)
AT[3] = new test(3)
K=0
for i=1 to 3
s[i]→ sorting (n,L,m,SL)
ST[i] = {m,SL}
testsort=AT[i]→ test(n,L,m,SL)
if testsort == true then
k=k+1
AST[k] = {m,SL}
end if
end for
```

```
if k=0 then
ran= random(1,3)
{m,SL} =ST[rand]
else if
{m,SL}= vc→ vote (AST,k)
endif
}
Class Voting_component
{
{int ,int} vote (in: AST,k)
{
if k==3 then
if AST[1]==AST[2] then
return AST[1]
else if AST[2]==AST[3] then
return AST[2]
else if AST[1]==AST[3] then
return AST[3]
endif
r=random(1,3)
return AST()
else if k==2 then // if 2 results pass their respective tests
if AST[1]==AST[2] then
return AST[1]
endif
ran=random (1,2)
return AST[r]
elseif k==1 then

return AST[1]

endif

}

Class Acceptence_test_1
boolean test(n, L[]; m, SL[]){
int count
For i=1 to m
If(SL[i]>0)
then,
for i=1 to m// if the elements in SL are not sorted
for j=i+1 to m-1
if (SL[i] < SL[j])
return true.
Else
Return false.
Else
Return false.
endif
Endfor
Endfor
}

Class Acceptence_test_2
boolean test( n, L[],m, SL[]){
int count
For i=1 to n
if(L[i]>=0) then
count++
end if
end for
}
If(m!=count)// if the number of positive integers in SL  is not equal to that of                    the positive
integers in L
return false
```

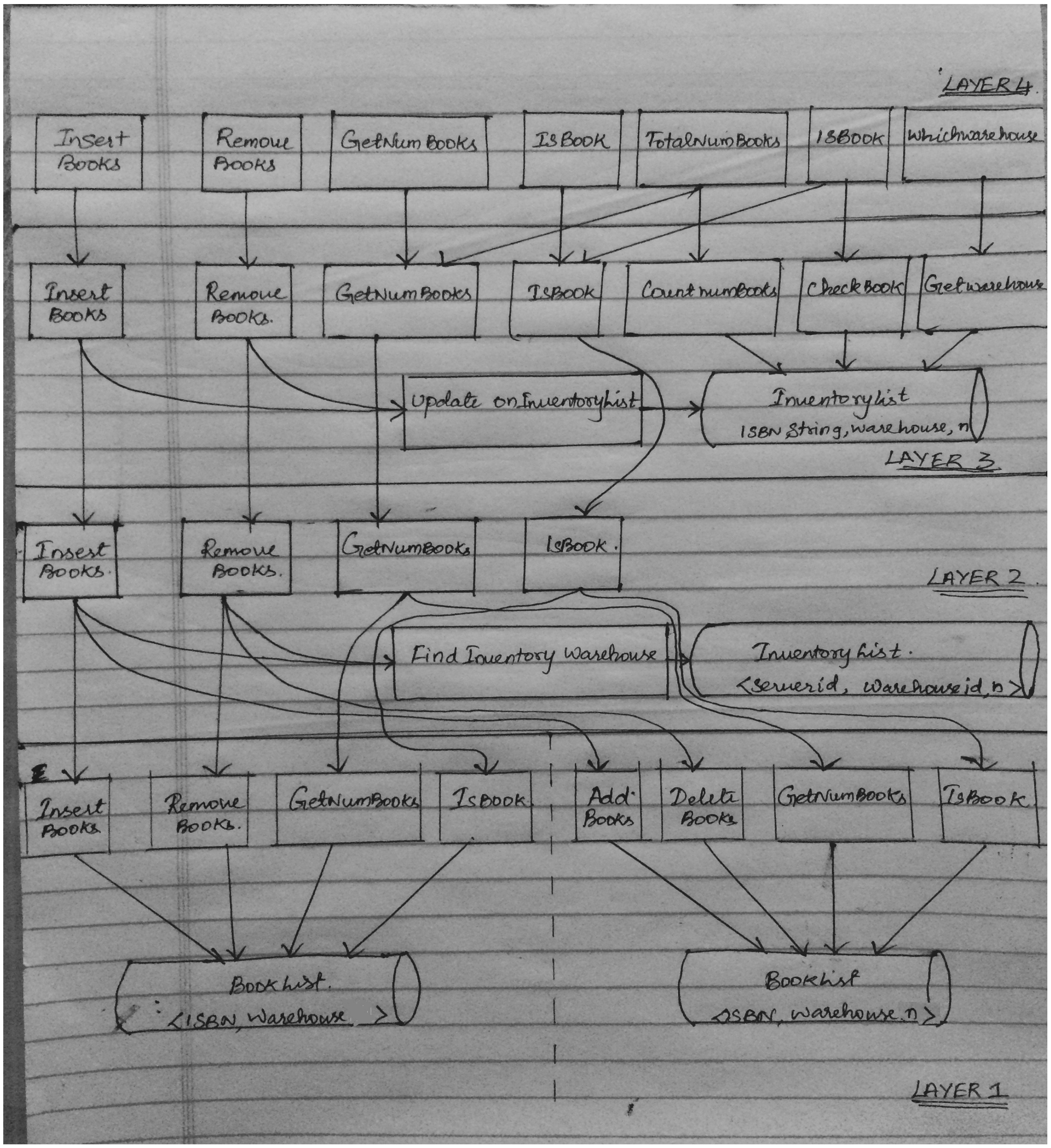
```
Class Acceptence_test_3:
boolean test(int n, L[],m, SL[])
{

for i=1 to 3 // if the list contains negative elements
if(SL[i] <0 )
return false
}
```

Sequence Diagram

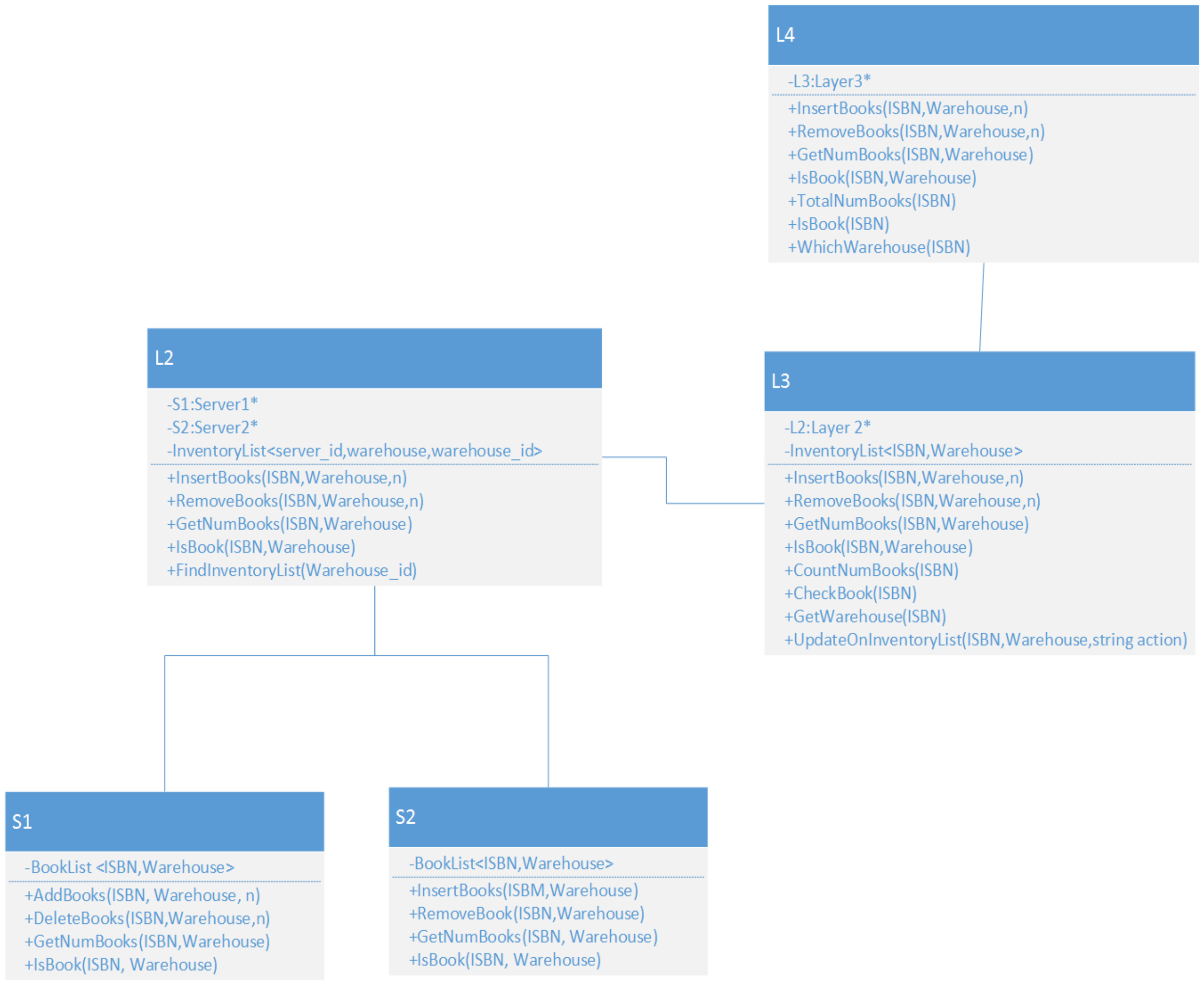


Problem #3

Strict Layered Architecture



Class Diagram


L4
-L3:Layer3*
+InsertBooks(ISBN,Warehouse,n)
+RemoveBooks(ISBN,Warehouse,n)
+GetNumBooks(ISBN,Warehouse)
+IsBook(ISBN,Warehouse)
+TotalNumBooks(ISBN)
+IsBook(ISBN)
+WhichWarehouse(ISBN)
L2
-S1:Server1*
-S2:Server2*
-InventoryList<server_id,warehouse,warehouse_id>
+InsertBooks(ISBN,Warehouse,n)
+RemoveBooks(ISBN,Warehouse,n)
+GetNumBooks(ISBN,Warehouse)
+IsBook(ISBN,Warehouse)
+FindInventoryList(Warehouse_id)
L3
-L2:Layer 2*
-InventoryList<ISBN,Warehouse>
+InsertBooks(ISBN,Warehouse,n)
+RemoveBooks(ISBN,Warehouse,n)
+GetNumBooks(ISBN,Warehouse)
+IsBook(ISBN,Warehouse)
+CountNumBooks(ISBN)
+CheckBook(ISBN)
+GetWarehouse(ISBN)
+UpdateOnInventoryList(ISBN,Warehouse,string action)
S1
-BookList <ISBN,Warehouse>
+AddBooks(ISBN, Warehouse, n)
+DeleteBooks(ISBN,Warehouse,n)
+GetNumBooks(ISBN,Warehouse)
+IsBook(ISBN, Warehouse)
S2
-BookList<ISBN,Warehouse>
+InsertBooks(ISBM,Warehouse)
+RemoveBook(ISBN,Warehouse)
+GetNumBooks(ISBN, Warehouse)
+IsBook(ISBN, Warehouse)

Pseudo-code

```
Layer 4
InsertBooks(ISBN,Warehouse,n)
{
L3→ InsertBooks(ISBN,Warehouse,n);
}

RemoveBooks(ISBN,Warehouse,n)
{
L3→ RemoveBooks(ISBN,Warehouse,n);
}

GetNumBooks(ISBN,Warehouse)
{
L3→ GetNumBooks(ISBN,Warehouse);
List the total number of books
}

IsBook(ISBN,Warehouse)
{
L3→ IsBook(ISBN,Warehouse);
}

TotalNumBooks(ISBN)
{
L3→ TotalNumBooks(ISBN);
Count the books available in all the warehouse
}

IsBook(ISBN)
{
boolean ans= L3→ IsBook(ISBN);
if(ans==true)
        L3→ TotalNumBooks(ISBN);
}

WhichWarehouse(ISBN)
{
L3→ Whichwarehouse(ISBN);
List the warehouse ID of the ISBN book
}

Layer 3
```

```
InsertBooks(ISBN,Warehouse,n)
{
update on InventoryList(ISBN,Warehouse,”in”)
L2→ InsertBooks(ISBN,Warehouse,n);
}

RemoveBooks(ISBN,Warehouse,n)
{
update on InventoryList(ISBN,Warehouse,”out”)
L2→ RemoveBooks(ISBN,Warehouse,n);
}

GetNumBooks(ISBN,Warehouse)
{
L2→ GetNumBooks(ISBN,Warehouse);
Return the count
}

IsBook(ISBN,Warehouse)
{
boolean ans= L2.IsBook(ISBN);
return ans;
}

CountNumBooks(ISBN)
{
onInventory List.update(ISBN,warehouse);
}

Checkbook(ISBN)
{
onInventory List.update(ISBN,warehouse);
}

GetWarehouse(ISBN)
{
go through the onInventory List, and return the warehouse
}

UpdateOnInventoryList(ISBN,Warehouse,string action)
{
if (action == “in”)   go through the InventoryList
if (ISBN exist in the OnInventory List)
        count the book and add to the totalnumbooks in the inventory list by n;
else if (action == “out”)
        go through the OnInventory List
        if (ISBN exist in OnInventory List)
        count the book and reduce the totalnumbooks in the inventory list by n;
      return;
}

Layer 2
InsertBook(ISBN,Warehouse_id,Warehouse,n)
{
int id= FindInventoryList(Warehouse_id)
if(id==1)
        server1.AddBook(ISBN,Warehouse,n);
else if(id==2)
        get n //n→ number of books that needed to be inserted
{
If n!= 0
{
        system2.InsertBook(ISBN,Warehouse);
        n=n--; // reduce the count of n by 1 until it becomes zero since the server 2 can add only one
book at a time
}
if n=0;
return;
}

RemoveBook(ISBN,Warehouse,Warehouse_id,n)
{
int id= FindInventoryList(Warehouse_id)
if(id==1)
        server1.DeleteBook(ISBN,Warehouse,n);
else if(id==2)
        get n→ number of books that needed to be removed
{
if n!=0
{
        system2.RemoveBook(ISBN,Warehouse);
        n=n--; //  reduce the count of n by 1 until it becomes zero since the server 2 can remove only
one book at a time
}
if n=0;
return;
}

GetNumBooks(ISBN,Warehouse)
{
Count List 1= Server1.GetNumBooks(ISBN,Warehouse)
Count List 2= Server2.GetnumBooks(ISBN,Warehouse)
combine Count List 1 and Count List 2 and return the combined count of warehouse ;
}

IsBook(ISBN,warehouse)
{
if(server1.IsBook(ISBN,Warehouse)==true)
        return true;
if(server2.IsBook(ISBN,warehouse)==true)
        return true;
return false
}

FindInventoryList(Warehouse_id)
{
go through the InventoryList
if(Warehouse_id is contained in Server 1)
        return 1;
else if(warehouse_id is contained in Server 2 )
        return 2;
}
```