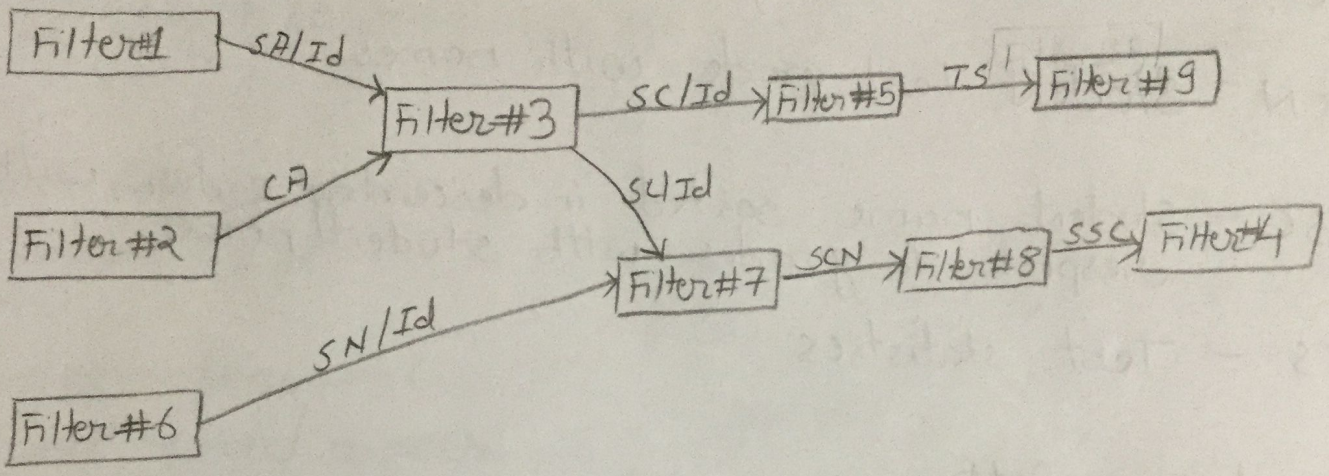
Assignment - 3

Name - Abhinav Pimpalgaonkar
Student Id - A20387324

Problem-1 Pipe and Filter Architecture

Part A



Filters.

Filter #1 - this filter read student test answer together with student ID's
Filter #2 - this filter read correct answer for the test
Filter #3 - this filter computes test grade with student ID
Filter #4 - this filter print test grades with names in the order as they are read from an input pipe
Filter #5 - this filter computes test statistics
Filter #6 - this filter read student name's together with student ID
Filter #7 - this filter merge student score with their names based on ID
Filter #8 - this filter report's test grade in descending order with student name's
Filter #9 - this filter report test statistics

Pipes

SA – Student test answer together with student Id

CA – Correct Answer for test

SN – Student Name together with their Id

SC – student test grade with student Id.

SCN – Student test grade with names

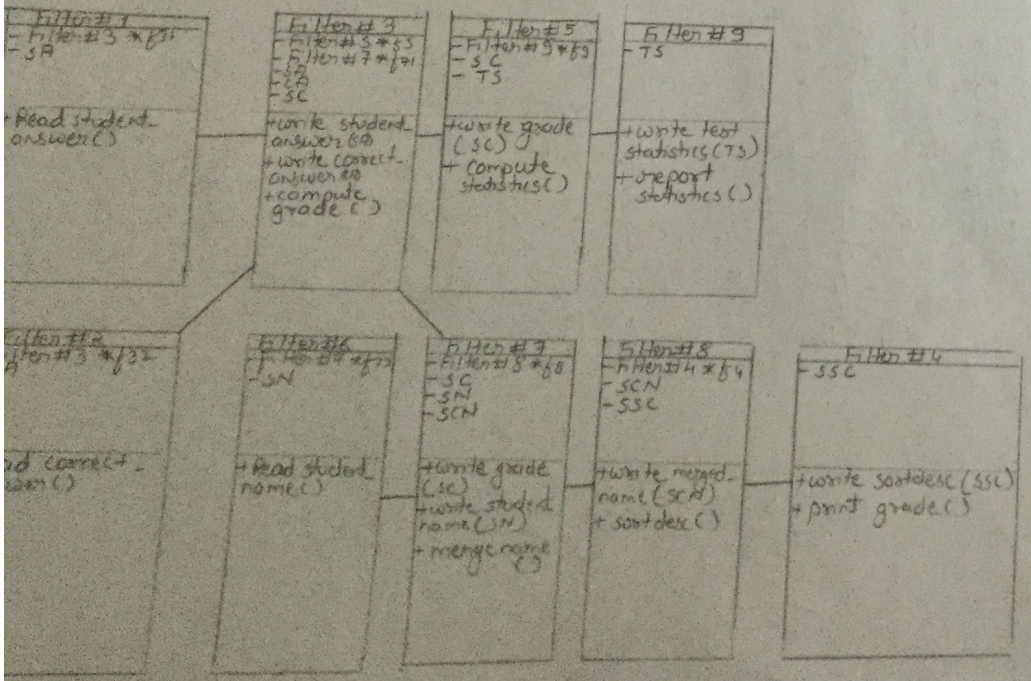
SSC – Student name sorted in descending order with respect to grades with student name

TS – Test statistics

Class diagram Part B

Pipe – unbuffered pipe

Filter – passive filters with push pipes



Pseudocode

① class Filter#1
Filter#3 *f31
SA

~~read~~ ReadStudentAnswer()
{ read student's answer together with student Id into SA;
f31 → writeStudentAnswer(SA);
}

② class Filter#2
Filter#3 *f32
CA
ReadCorrectAnswer()
{ read correct answer into CA;
f32 → writeCorrectAnswer(CA);
}

③ class Filter#6
Filter#7 *f72
SN
ReadStudentName()
{ read student's name together with student ID into SN;
f72 → writeStudentName(SN);
}

④ class Filter#3
Filter #5 *f5
Filter #7 *f71
SA
CA
SC
int SAF = 0; // Flag to check if SA received
int CAF = 0; // Flag to check if CA received
writeStudentAnswer(SA)
{ store in SA;
IF (CAF == 1)
{ computeGrade();
SAF = 0;
CAF = 0;
} Else SAF = 1;
}
writeCorrectAnswer(CA)
{ store in CA;
IF (SAF == 1)
{ computeGrade()
SAF = 0;
CAF = 0;
}
Else CAF = 1
}
computeGrade()
{ SC = computeStudentGrades(SA, CA);
f5 → writeGrade(SC);
f71 → writeGrade(SC);
}
computeStudentGrade(SA, CA)
{ // private function to compute // grades
~~Star~~ Compute grades and store in SC
}

⑤ class Filter #5
Filter #9 *f9
SC
TS
write grade (SC)
{ store in SC the grades;
compute statistics ()
}
compute statistics ()
{ TS = compute stats (SC)
f9 → write statistics (TS)
}
compute stats (SC)
{ compute statistics based
on grades and store
in TS;
}
Note: compute stats (SC) is private function

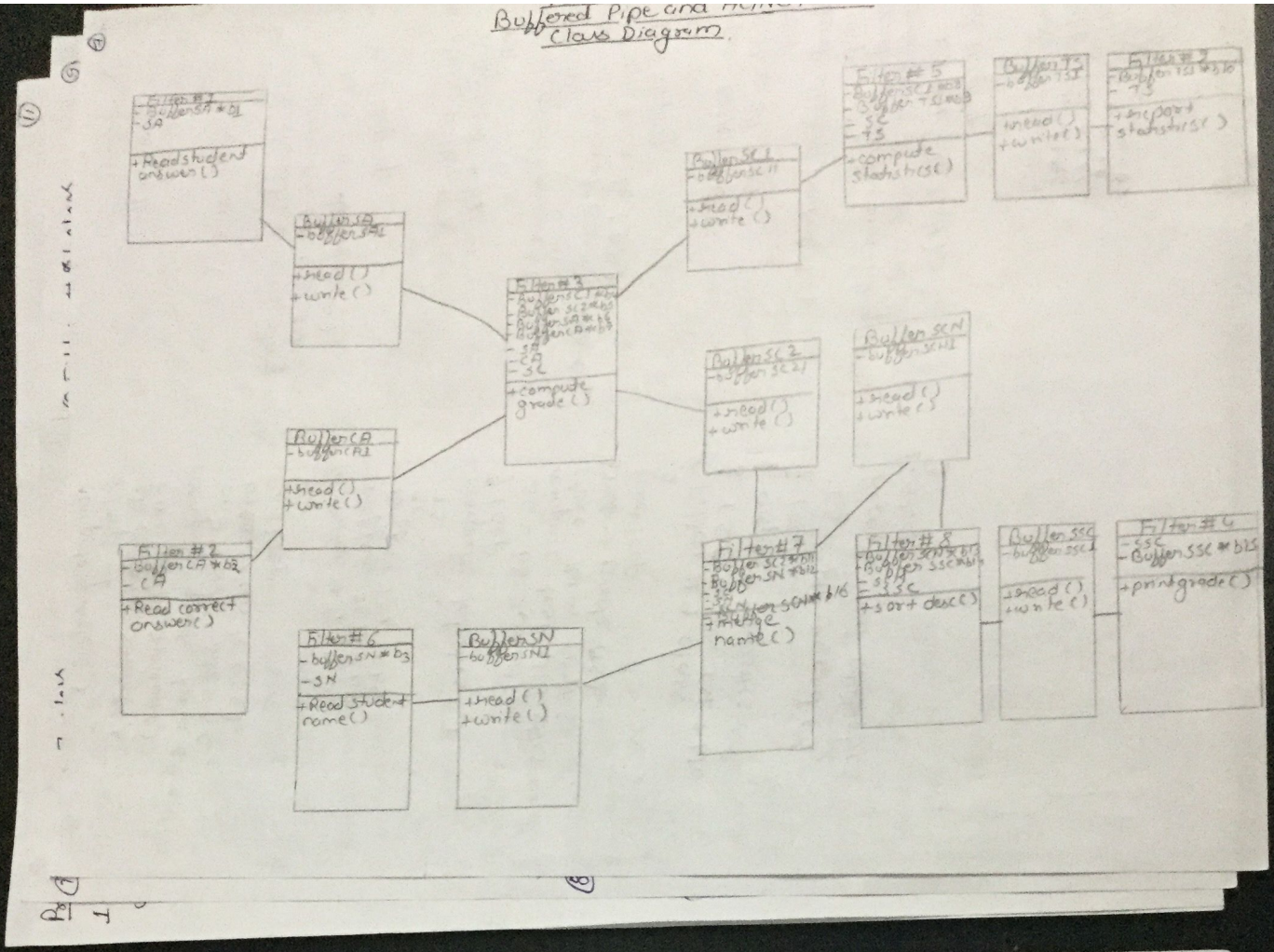
⑥ class Filter #9
TS
write test statistics (TS)
{ store in TS;
report statistics ();
}
report statistics ()
{ It will provide/print
report of statistics
}

⑦ class Filter #7
Filter #8 *f8
SC
SN
SCN
~~write grade~~
SCF=0 : // Flag to check if grades received
SNF=0 : // Flag to check if names received.
write grade (SC)
{ store in SC;
IF (SCN == 1)
{ merge name ();
SCF=0;
SNF=0;
}
Else SCF=1;
}
write student name (SN)
{ store in SN;
IF (SNF == 1)
{ merge name ();
SCF=0;
SNF=0;
}
Else SNF=1;
}
merge name ()
{ SCN = merge name and grades (SC, SN);
f8 → write merged name (SCN);
}
merge name and grades (SC, SN)
{ merge student name with
grades based on student
ID and store in SCN;
// Private function
}

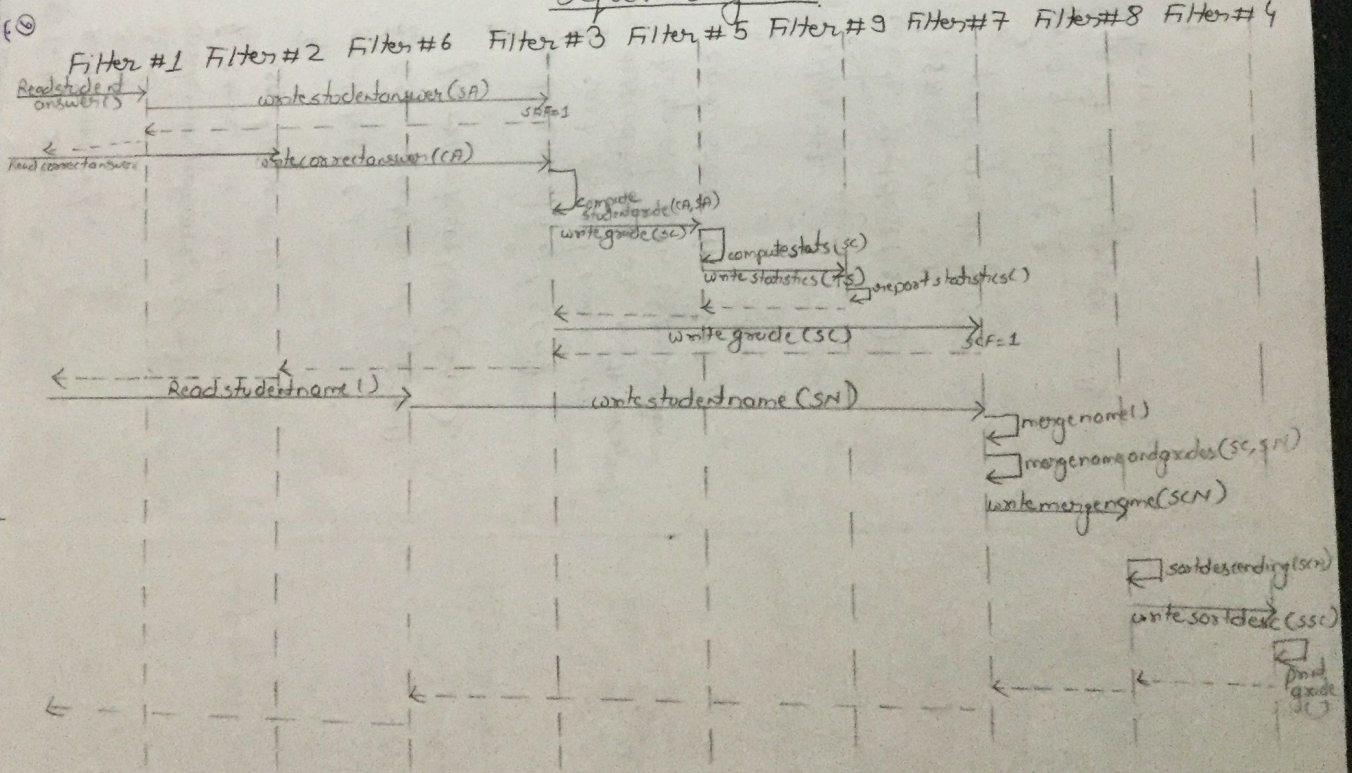
⑧ Filter #8
Filter #4 *f4
SCN
SSC
write merged name (SCN)
{ store in SCN
sort desc ();
}
sort desc ()
{ SSC = sort descending (SCN)
f4 → write sort desc (SSC);
}
sort descending (SCN)
{ sort student name in
descending order with respect
to grades with student name
and store in SSC;
}

⑨ Filter #4
SSC
write sort desc (SSC)
{ store in SSC
print grade ();
}
print grade ()
{ print test grades with
names
}

Buffered Pipe and Filter
Class Diagram



Sequence Diagram



Pseudocode

① class Filter #1
Buffer SA *b1
SA
Read studentanswer ()
{ Loop
read student's answer along with the student's Id into SA;
b1 → write ~~studentans~~ (SA);
End looping
}

② class Filter #2
Buffer CA *b2
CA
Read correctanswer ()
{ Loop
read correct answer into CA;
b2 → write (CA);
End looping
}

class Filter #6
Buffer SN *b3
SN
Read studentname ()
{ loop
read student name along with student's Id into SN;
b3 → write (SN);
nd looping

lass Filter #3
uffer SA *b6
uffer CA *b7
uffer SC1 *b4
uffer SC2 *b5
ob
}

computegrade ()
{ Loop
SA = b6 → read ();
CA = b7 → read ();
compute grade from SA and CA and store into SC;
b4 → write (sc);
End loop
}

⑤ Filter #5 class
Buffer SC1 *b8
Buffer TSL *b9
SC
TS
compute statistics ()
{ Loop
SC = b8 → read ();
compute test statistics and store them in TS
b9 → write (TS)
End loop
}

⑥ Filter #9 class
{ TS; Buffer TSL *b10
{ report statistics ()
{ Loop
TS = b10 → read ();
Report test statistics
End loop
}

① Filter #7 class
Buffer SC2 *b11
Buffer SN *b12
SC
SN
SCN
Buffer SCN *b16
mergename ()
{ Loop
SC = b11 → read ();
SN = b12 → read ();
Merge name with grades based on student Id's; and store in SCN;
b16 → write (SCN);
End Loop
}

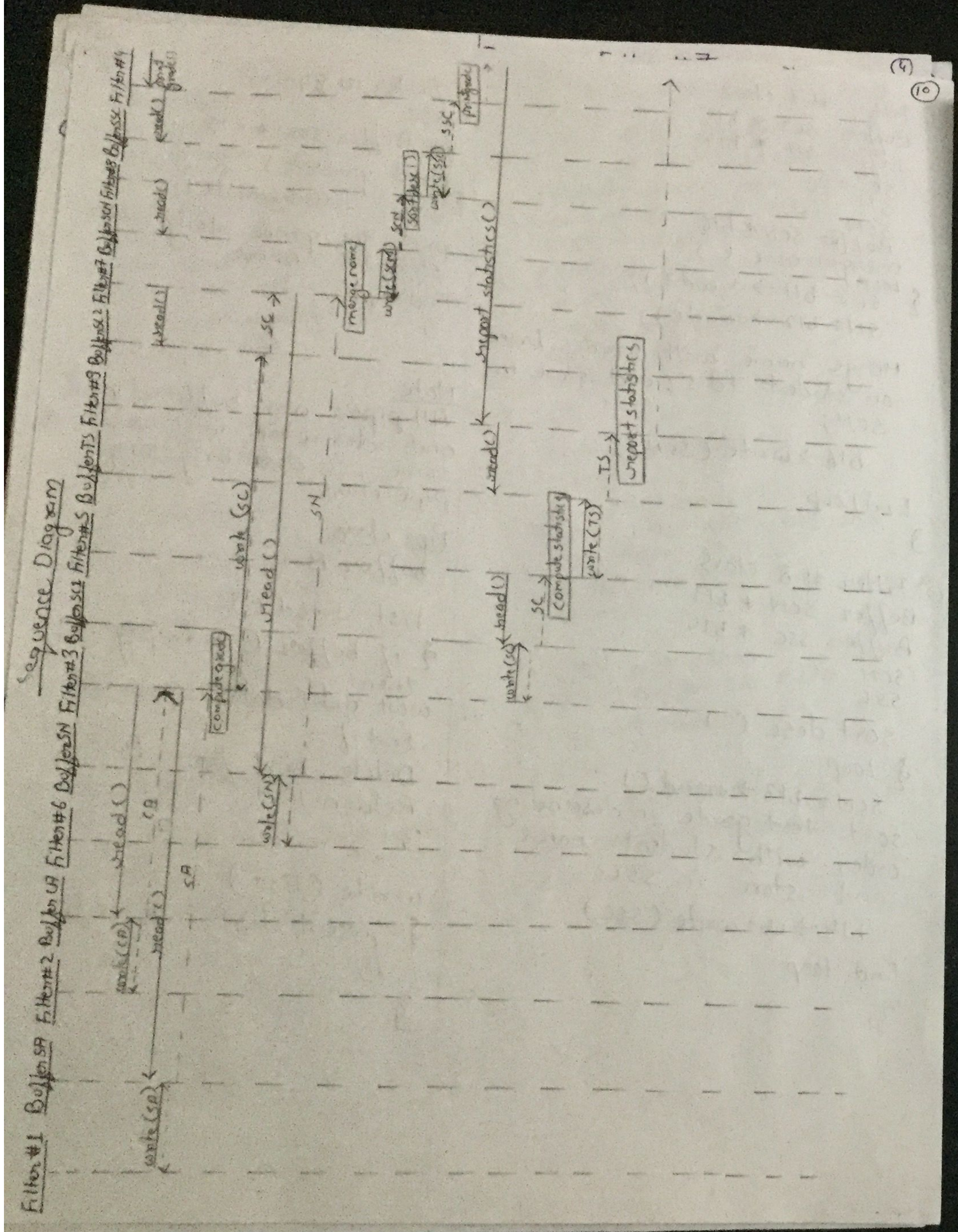
⑧ Filter #8 class
Buffer SCN *b13
Buffer SSC *b14
SCN
SSC
sort desc ()
{ Loop
SCN = b13 → read ()
sort test grade in descending order with student's name and store in ssc
b14 → write (ssc)
End loop
}

⑨ Filter #4 class
SSC
Buffer SSC *b15
print grade ()
{ Loop
SSC = b15 → read ();
print the grade along with student's name;
End loop
}

Note
All pipe's are buffered pipe and description will be same so describing single pipe class.

Pipe class
buffer B
list read ()
{ if buffer B is empty then wait until data is received
End if
Delete list from B
Return list
}

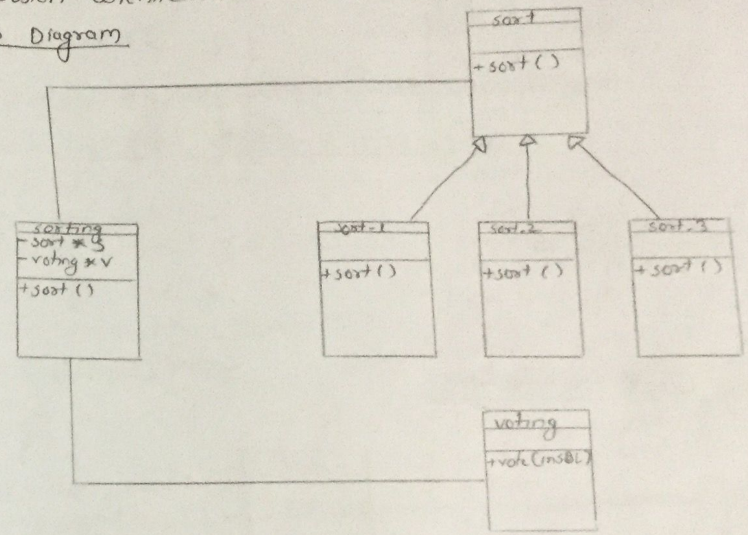
write (list)
{ insert list into buffer B
}



Problem 2 Fault Tolerant

1. N-version architecture

(a) Class Diagram



Pseudocode

① class sorting

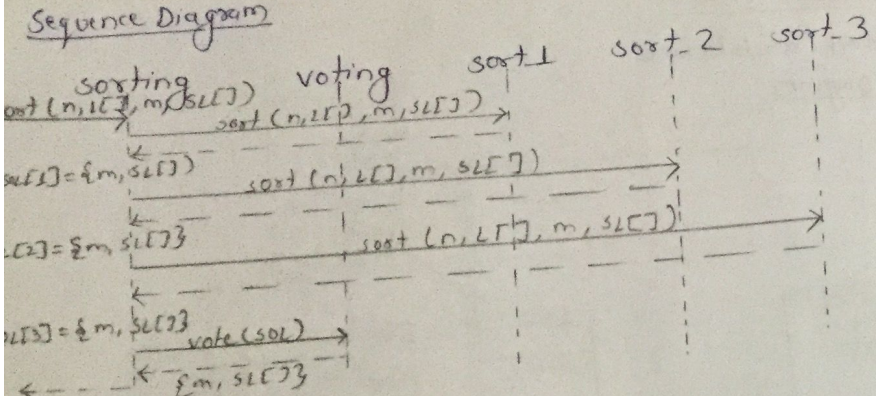
```
sort *S
voting *V
void sort(in int n, int L[]; out int m, int SL[])
{ SL[] is a list {[redacted] int, int[]}
  sorting S[]
    S[1] = new_sort_1()
    S[2] = new_sort_2()
    S[3] = new_sort_3()
  for i = 1 to 3
    S[i] -> sort(n, L[], [redacted] m, SL[])
    SL[i] = {m, SL[]}
  End for
  {m, SL[]} = V -> vote(SL)
}
```

② class voting

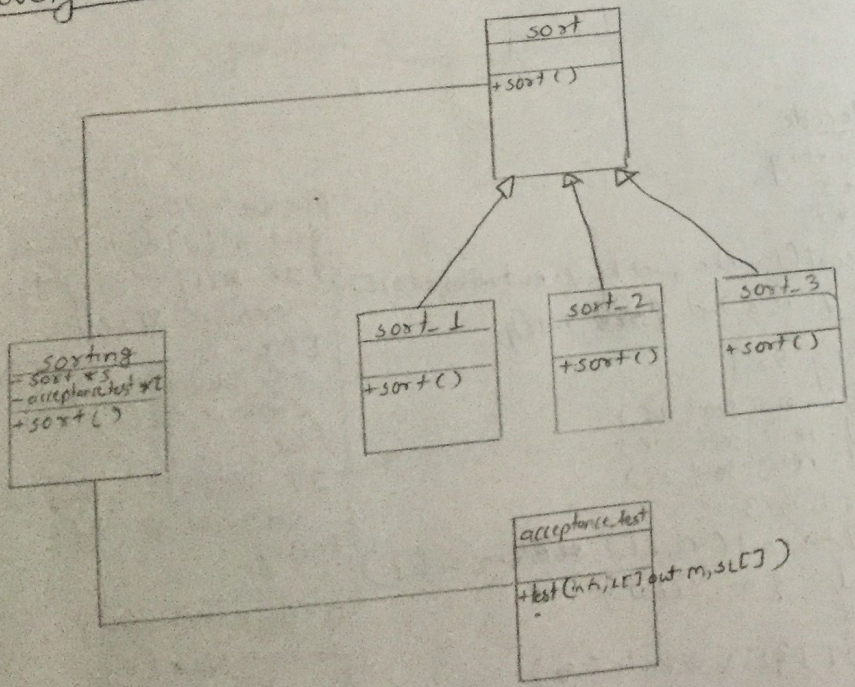
```
{int, int[]} vote (in SL)
{ IF SL[1] == SL[2] then
    return SL[1]
  Else
    IF SL[2] == SL[3] then
    return SL[2]
  Else
    IF SL[3] == SL[1] then
    return SL[3]
  End if
  a = GenerateRandom (1,3)
  return SL[a]
}
```

Note: Generat Random will generate random number b/w 1-3

Sequence Diagram



③ Recovery Block Architecture



Pseudocode

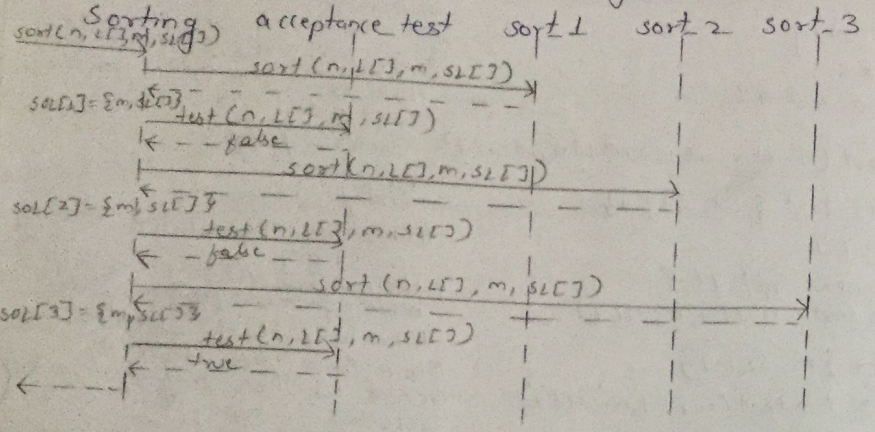
```
class sorting
  sort *s
  acceptance_test *t

Void sort (in int n, int L[], out int m, int SL[])
{ SOL is List {int, int[]}
  Sort s[]
  S[1] = new_sort_1()
  S[1] -> sort (n, L[], m, SL[])
  SOL[1] = {m, SL[]}
  result = t -> test (n, L[], m, SL[])
  if (result == 1) then
  exit;
  End if
  S[2] = new_sort_2()
  S[2] -> sort (n, L[], m, SL[])
  SOL[2] = {m, SL[]}
  result = t -> test (n, L[], m, SL[])
  if (result == 1) then
  exit;
  End if
  S[3] = new_sort_3()
  S[3] -> sort (n, L[], m, SL[])
  SOL[3] = {m, SL[]}
  result = t -> test (n, L[], m, SL[])
  if (result == 1) then
  exit;
  end if
// If all test fails generate random number
// and randomly select from sorted list SOL
  a = Generate random (1, 3)
  {m, SL[]} = SOL[a]
}
```

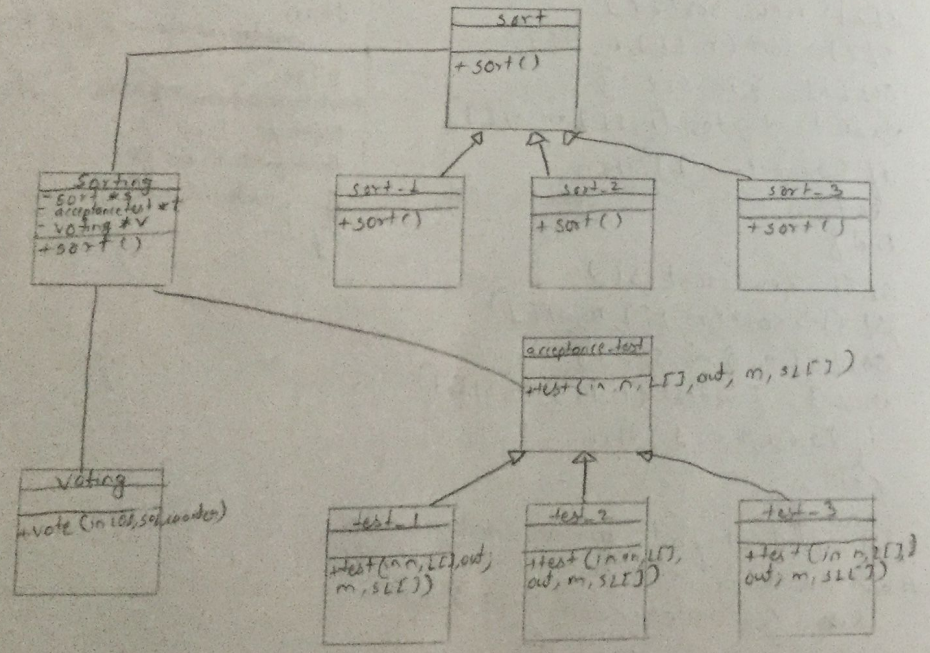
class acceptance test

```
boolean test (in int n, int L[],
out int m, int SL[])
{ if (m > n) then
  // size of output list is greater
  than original list
  return false,
  et
  end if
// checking the sorted array
" in SL[]
bool compare sort [m] = {false};
~~bool compare sort [m] = {false};~~
// This will check whether the
// two consecutive integer to
// check whether they are sorted
// Starting for loop with so as to
// compare ~~0~~ 2nd element with 1st element
// ~~sort (SL[])~~ // sort in Ascending order
  For i=1 to n
  If compare sort [SL[i]] >=
     compare sort [SL[i-1]]
  then
  ~~return true~~ int b=1;
  else
  ~~bool b = false~~; return false;
  ~~End if~~ End if
  ~~End for~~ End for
  } return true;
  }
```

Sequence Diagram



3 N Self Checking Architecture



Pseudocode

Class sorting

```
sort *s
acceptance test *t
voting *v
void sort(in int n, int L[], out; int m, int SL[])
{ SOL is a {int, int[]}, list
  LOS is a {int, int[]}, list
  sort S[]
  acceptance_test +[]
  S[1] = new_sort_1()
  +[1] = new_test_1()
  S[2] = new_sort_2()
  +[2] = new_test_2()
  S[3] = new_sort_3()
  t[3] = new_test_3()
int counter = 0
For i = 1 to 3
  S[i] → sort(n, L[], m, SL[])
  SOL[i] = {m, SL[]}
  result = t[i] → test(n, L[], m, SL[])
  If result == true Then
     counter = counter + 1
     LOS[i] = {m, SL[]}
  End if
End for
  {m, SL[]} = v → vote(LOS, SOL, counter)
}
```

Class voting

```
{int, int[]} vote(in LOS, SOL, counter)
{
If counter == 0 then
// no result accepted
a = RandomGenerate(1,3)
return SOL[a]
End if
Else if counter == 1 then
  return LOS[1] // result accept
End if
Else if counter == 2 then
  if LOS[1] == LOS[2] then
     return LOS[1]
End if
  r = generaterandom(1,2)
  return LOS[r]
End if
// Two result accepted case above
Else if counter == 3 then
// Three result accepted.
  if LOS[1] == LOS[2] then
     return LOS[1]
End if
  else if LOS[2] == LOS[3] then
     return LOS[2]
End if
  else if LOS[3] == LOS[1] then
     return LOS[3]
End If
  r = generaterandom(1,3)
  return LOS[r]
End If
}
```

③ class test_1:

```
boolean ~~sort~~ test(in int n, int L[], out int m, int SL[])
{ if m>n then
    return false
  End if
  ~~bool sort test 1 [m] = false~~
  bool sort test 1 [m] = false;
  For i = 1 to m
    If bool sort test 1 [SL[i]] >=
       bool sort test 1 [SL[i-1]] then
       ~~int b=1;~~ bool sort test 1 [SL[i]]
                    = true;
    Else
       return false;
    End if
  End for
  return true;
}
```

④ class test_2;

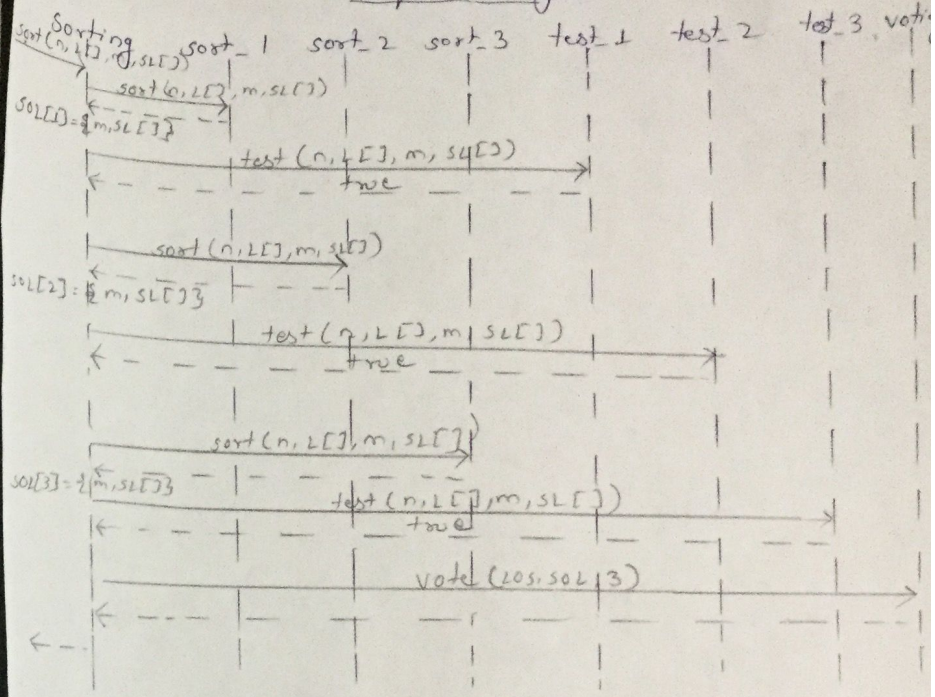
```
boolean test (in int n, int L[], out; int m, int SL[])
 { if m>n then
     return false
   End if
   ~~bool sort test 2 [m] = false~~
   bool sort test 2 [m] = false;
   For j=1 to m
     For i=0 to j
IF bool sort test 2 [SL[j]] >=
   bool sort test 2 [SL[i]]
     ~~int b=1;~~ bool sort test 2 [SL[j]] = true
   Else
     return false
End IF
End for
End for
return true;
}.
```

⑤ class test_3:

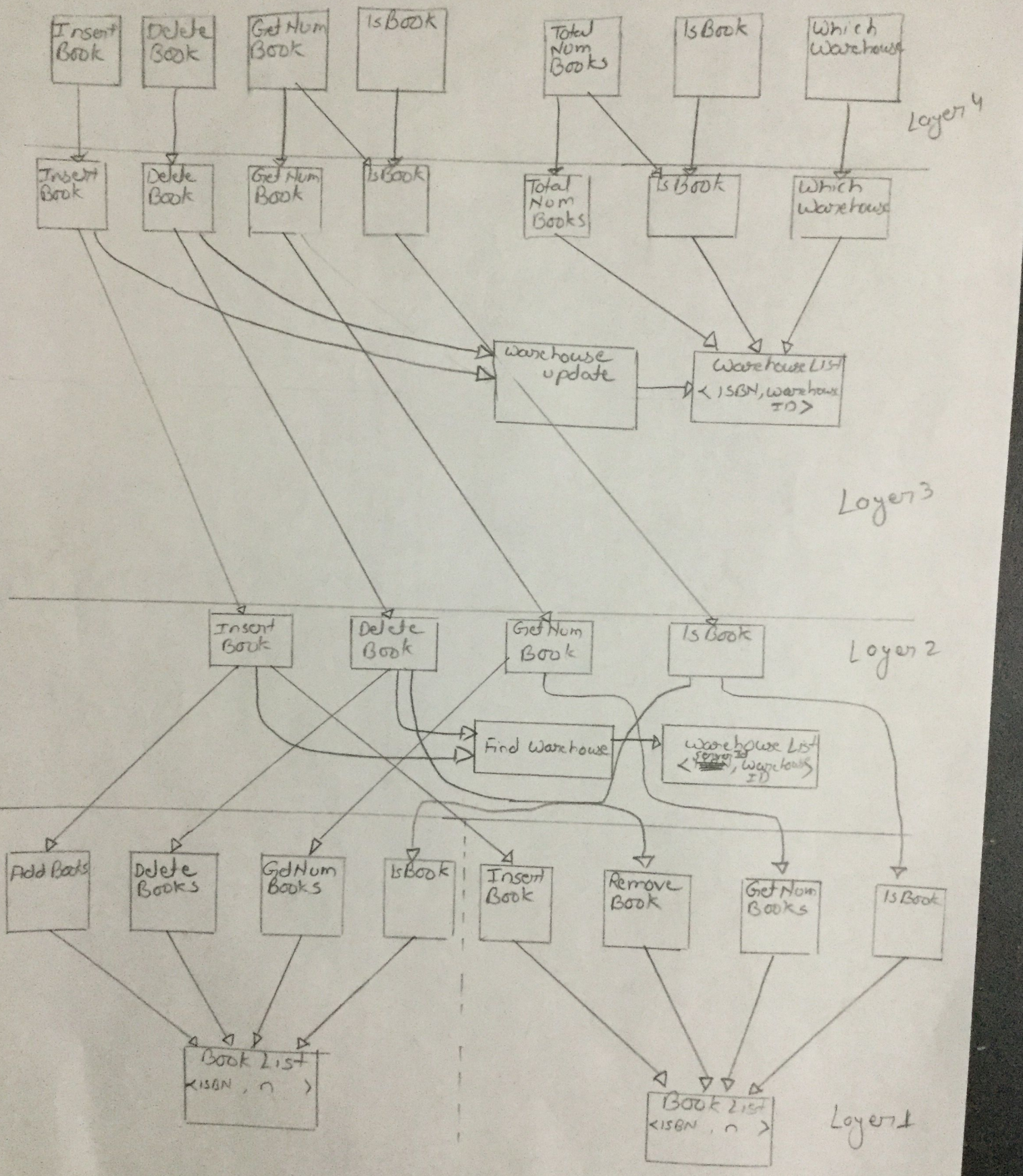
```
boolean test (int n, int L[], out, int m, int SL[])
{ if m>n then
    return false
  End if
  ~~bool sort test 3 [m] =~~ bool sort test 3 [m]
                          = false;
  Reverse sort (SL[])
  // list sorted in descending order
  For i: 0 to m-1
If bool sort test 3 [SL[i+1]]
   < ~~SL[i]~~ bool sort test 3 [SL[i]]
  bool sort test 3 [SL[i]] = true;
  else
  return false;
End if
 End for
 return true;
}.
```

Note: All the acceptance test condition will be different with different way of testing.

Sequence Diagram



Q3 Strict Layered Architecture.



Pseudocode

```
0 class L4
  void insertBook(ISBN string, warehouse int, n int)
  { L3 → insertBook(ISBN, warehouse, n)
  }
  Void DeleteBook(ISBN string, warehouse int, n int)
  { L3 → DeleteBooks(ISBN, warehouse, n)
  }
int ~~void~~ GetNumBooks(ISBN string, warehouse int)
  { int x; IF(L3 → IsBook(ISBN, int) == 1)
    L3 → GetNumBooks(ISBN, warehouse)
    return x; else return 0; Endif
  }
3.
~~void~~ int IsBook(ISBN string, warehouse int)
  { int a;
    a = L3 → IsBook(ISBN, warehouse)
    return a;
  }
  int TotalNumBooks(ISBN string)
  { int b;
    boolean result = L3 → IsBook(ISBN)
    IF result == true then
    b = L3 → TotalNumBooks(ISBN)
    Endif
    return b;
  }
  int isBook(ISBN string)
  { int c;
    c = L3 → isBook(ISBN)
    return c;
  }
```

```
int whichWarehouse(ISBN string)
{ int d = L3→ whichWarehouse(ISBN)
  return d;
}.
```

② class L3

```
void InsertBook (ISBN string, warehouse int, n int)
{ warehouse update (ISBN, warehouse_id, "in")
      L2 → InsertBook (ISBN, warehouse, n)
}

void DeleteBook (ISBN string, warehouse int, n int)
{ warehouse update (ISBN, warehouse, "out")
      L2 → DeleteBook (ISBN, warehouse, n)
}

int GetNumBooks (ISBN string, warehouse int)
{ int y = L2 → GetNumBooks (ISBN, warehouse)
  return y;
}

int IsBook (ISBN string, warehouse int)
{ int z = L2 → IsBook (ISBN, warehouse)
  return z;
}

int TotalNumBooks (ISBN string)
{ get total number of books from warehouse
  list based on ISBN
}

int IsBook (ISBN string)
{ Check the warehouse list for the book
  based on ISBN
}

which Warehouse (ISBN) ~~warehouse, action~~)
{ ~~If (action == "in")~~ // Find the warehouse
  number for the ISBN provided
}
```



```
warehouse update (ISBN, warehouseID, action)
{ If (action == "in")
    Warehouse list.add new ISBN, warehouseId

  Else
    If (action == "out")
    // Find the book based on ISBN by traversing the
    // whole list
      Warehous list.remove ISBN, warehouse ID

    Else
      // Error Message
    End if
  End if
}.
```

③ class L2

```
void insertBook (ISBN string, warehouse int, n int)
{ IF (n>0) then
  //check the signature of function
    S11 → AddBooks (ISBN, warehouse, n)
  Else
    S12 → InsertBook (ISBN, warehouse)
  End If
} { int id = Find Warehouse (warehouse)
    if (id == 1) and n>0)
    S11 → AddBooks (ISBN, warehouse, n)
    else if (id == 2) and n>0
      S12 → InsertBook (ISBN, warehouse)

}
```

```
void DeleteBook (ISBN string, warehouse int, n int)
{ int id1 = Find Warehouse (warehouse)
   If id1 == 1 and n>0 then
     S11 → DeleteBooks (ISBN, warehouse, n)
   else
   If id1 == 2 and n>0 then
     S12 → Remove ~~Delete~~ Books (ISBN, warehouse)
   End if
  End if
}
int GetNumBooks (ISBN string, warehouse int)
{ int a = ~~GetNumBooks~~ S11 → GetNumBooks (ISBN, warehouse)
   int b = ~~GetNumBooks~~ S12 → GetNumBooks (ISBN, warehouse)
    add a and b and return result
}
int IsBook (ISBN string, warehouse int)
{ If S11 → isBook (ISBN, warehouse) == true then
    return 1;
   else
   if S12 → isBook (ISBN, warehouse) == true then
    return true;
   else
   // Print Does not exist
  End if
}
```



```
Find warehouse (warehouse ID)
{ Check in warehouse list
  If (found in server 1)
   return 1;
  else
  If (found in server 2
   return 2;
}
```

④
```
class Server S1
void Add Books (ISBN string, warehouse int, n int)
{ Add the number of books n in the book list with ISBN
}
void DeleteBooks (ISBN string, warehouse int, n int)
{ Delete the number of book n from book list with ISBN}
}
int GetNumBooks (ISBN string, warehouse int)
{ return the total number of books of an ISBN
}
int isBook (ISBN string, warehouse int)
{ return 1, if ISBN book exist otherwise return 0
}
```

⑤
```
class Server S2
void InsertBrok (ISBN string, warehouse int)
{ Add a book to book list based on ISBN.
}
void Remove Book (ISBN string, warehouse int)
{ Delete a book from book list based on ISBN
}
int GetNumBooks (ISBN string, warehouse int)
{ return total number of books n based on ISBN from
  book list
}
int isBook (ISBN string, warehouse)
{ return 1 if ISBN book exist in book list else
        return 0;
}
```