PROBLEM #1

Consider the problem of designing a system using **Pipes and Filters** architecture. The system should provide the following functionality:

- Read student's test answers together with student's IDs.
- Read student's names together with their IDs.
- Read correct answers for the test.
- Compute test scores.
- Compute test statistics: a mean and a standard deviation
- Report test's scores in a **descending** order with respect to scores with student names
- Report test statistics: a mean and a standard deviation

It was decided to use a Pipe and Filter architecture using the existing filters. The following existing filters are available:

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 reads student's names together with their IDs.
Filter #4: this filter computes test scores.
Filter #5: this filter prints test scores with student names in the order as they are read from an input pipe.

Part A:

Provide the Pipe and Filter architecture for the Grader system. In your design you should use all existing filters. If necessary, introduce additional Filters in your design and describe their responsibilities. Show your Pipe and Filter architecture as a directed graph consisting of Filters as nodes and Pipes as edges in the graph.

Part B:

1. For the Pipe and Filter architecture of Part A, assume that each pipe is an **un-buffered** pipe and all filters are **passive** filters with **pull-out** pipes. However, **Filter #4** is the only filter with **push** pipes. Notice that **all** new filters should be with **pull-out** pipes.
2. Use object-oriented design to refine your design. Each filter should be represented by a class. Provide a class diagram for your design. For each class identify operations supported by the class and its attributes. Describe each operation using pseudo-code.
3. Provide a sequence diagram for a typical execution of the system based on the class diagram of Step 2.

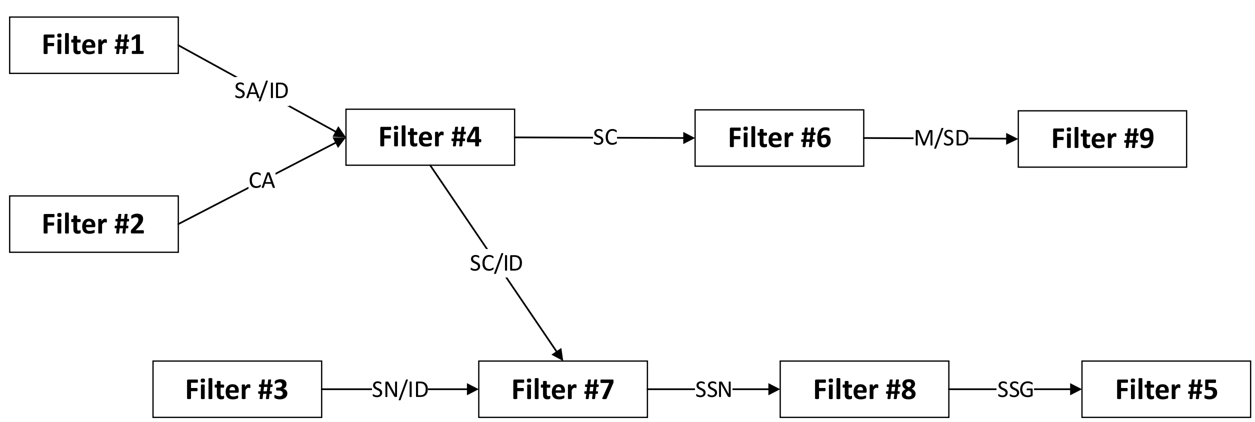
Part A

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 reads student’s names together with their IDs.
Filter #4: this filter computes test scores.
Filter #5: this filter prints test scores with student names in the order as they are read from an input pipe.
Filter #6: this filter computes the mean and the standard deviation.
Filter #7: this filter merges scores with name and ID.
Filter #8: this filter sort student names and IDs in a descending grade order with respect to test scores.
Filter #9: this filter reports test statistics.

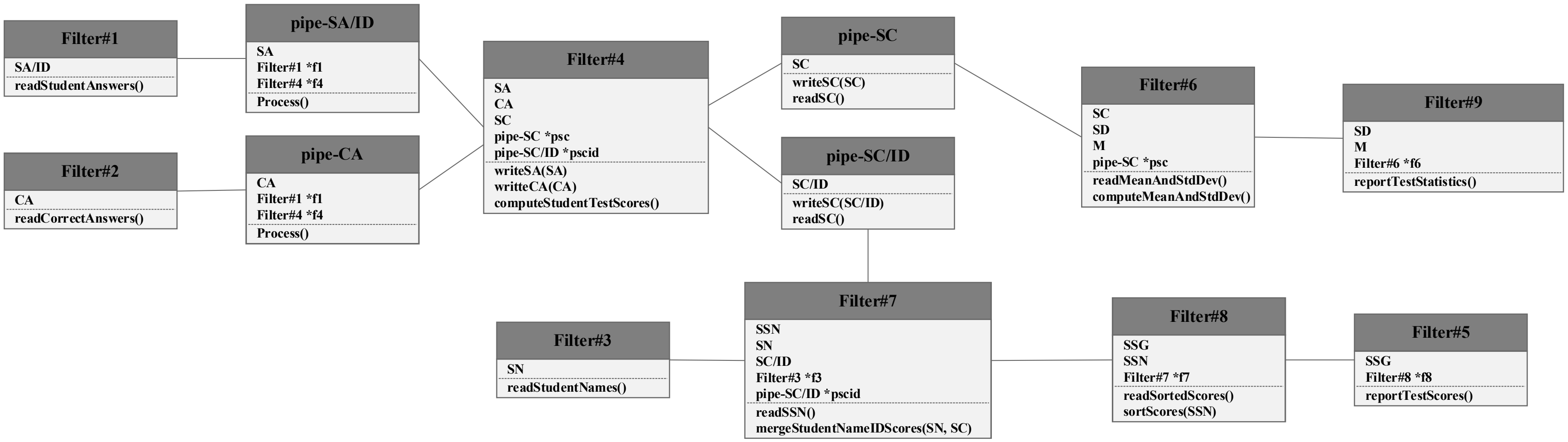
Pipes

SN: student’s names together with student's ID
SA: student's test answers
CA: correct answers for the test
SC: student's test scores
M: mean
SD: standard deviation
SSN: student names, ID and test scores
SSG student names, IDs and test scores sorted in a descending grade order



Part B

Class Diagram



Class Filter #1

SA // student's test answers together with student's ID

```
readStudentAnswers (){
        reads student's test answers together with student's IDs into SA
        return SA
}
```

Class Filter #2

CA //correct answers for the test

```
readCorrectAnswers (){
        read correct answers into CA
        return CA
}
```

Class Filter #3

SN // student's names together with student's ID

```
readStudentNames (){
        reads student's names with their IDs into SN
        return SN
}
```

Class Filter #4

```
SA // student's test answers together with student's ID
CA // correct answers for the test
SC // student's test scores
pipe-SC *psc
pipe-SC/ID *pscid
flagCA = false
flagSA = false

writeSA(SA){
    store into SA
    If flagCA == true Then
        computeStudentTestScores(CA,SA)
        flagCA = false
```

```
        flagSA = false
      Else  flagSA = true
    End if
}

writeCA(CA){
    store into CA
    If flagSA == true Then
        computeStudentTestScores(CA,SA)
        flagSA = false
        flagCA = false
    Else
        flagCA = true
    End if
}

computeStudentTestScores(CA,SA){
    grade student answers with student ID into SC
    psc->writeSC(SC)
    pscid->writeSC(SC/ID)
}
```

Class Filter 5

```
SSG // student names, IDs and test scores sorted in a descending grade order
Filter#8 *f8

reportTestScores(){
        SSG = f8->readSortedScores()
        print SSG
}
```

Class Filter #6

```
SC  // student's test scores
M  // mean
SD // standard deviation
pipe-SC *psc

readMeanAndStdDev(){
          SC= psc->readSC()
          If SC !=NULL Then
              computeMeanAndStdDev(SC)
              return {M, SD}
```

```
        Else
            return NULL
        End If
}

computeMeanAndStdDev(SC){
        compute the mean with SC and store into M
        compute the standard deviation with M and SC and store into SD
}
```

Class Filter #7

```
SSN // student names, ID and test scores
SN // student's names together with student's ID
SC/ID  // student's test scores together with student's ID

Filter#3 *f3
pipe-SC/ID *pscid

readSSN(){
        SN = f3->readStudentNames()
        SC/ID = pscid ->readSC()
        If SC/ID != NULL Then
                SSN = mergeStudentNameIDScores(SN, SC)
                return SSN
        Else
                return NULL
        End if

}

mergeStudentNameIDScores(SN, SC){
        merge the student's names, IDs and test scores and return merged list
}
```

Class Filter #8

```
SSG // student names, IDs and test scores sorted in a descending grade order
SSN // student names, ID and test scores
Filter#7 *f7

readSortedScores(){
        SSN = f7->readSSN()
        If SSN != NULL Then
            SSG = sortScores(SSN)
            return SSG
```

```
        Else
            return NULL
        End if
}


sortScores(SSN){
        sort student name, ID and scores in a descending grade order
}
```

Class Filter #9

```
M // mean
SD // standard deviation
Filter#6 *f6

reportTestStatistics(){
        {M, SD} = p6-> readMeanAndStdDev()
        If {M, SD} != NULL Then
            print M and SD
        End if
}
```

Class pipe-SA/ID

```
SA  // student's test answers together with student's ID
Filter#1 *f1
Filter#4 *f4

Process(){
    SA = f1->readStudentAnswers() //blocked until receive SA
    f4-> writeSA(SA)
}
```

Class pipe-CA

```
CA  //correct answers for the test
Filter#1 *f1
Filter#4 *f4

Process(){
    CA = f1-> readCorrectAnswers() //blocked until receive CA
    f4-> writeCA(CA)
}
```

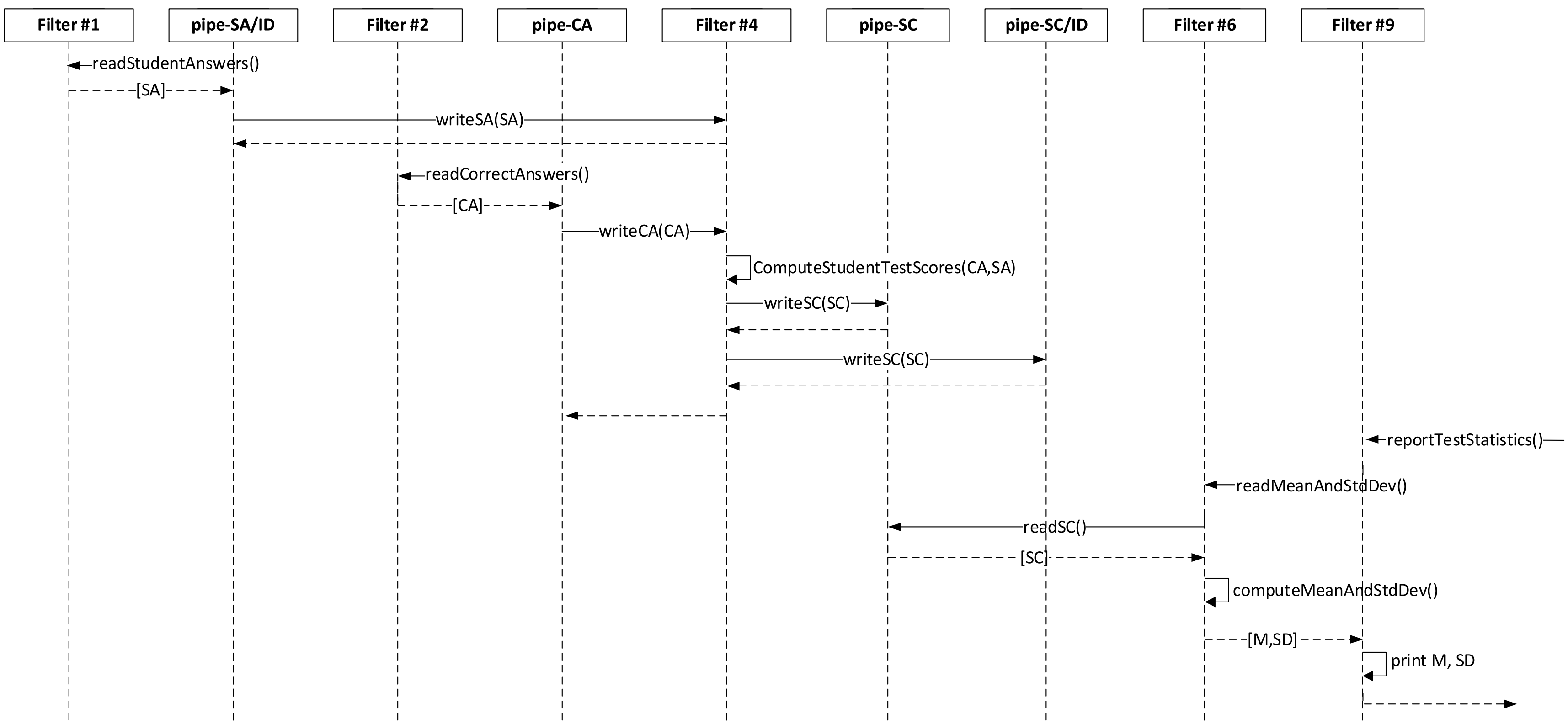
Class pipe-SC/ID
SC // student's test scores

```
writeSC(SC){
      store into SC
}

readSC(){
       If SC != NULL Then
           return SC
       Else
           return NULL
        End if
}
```

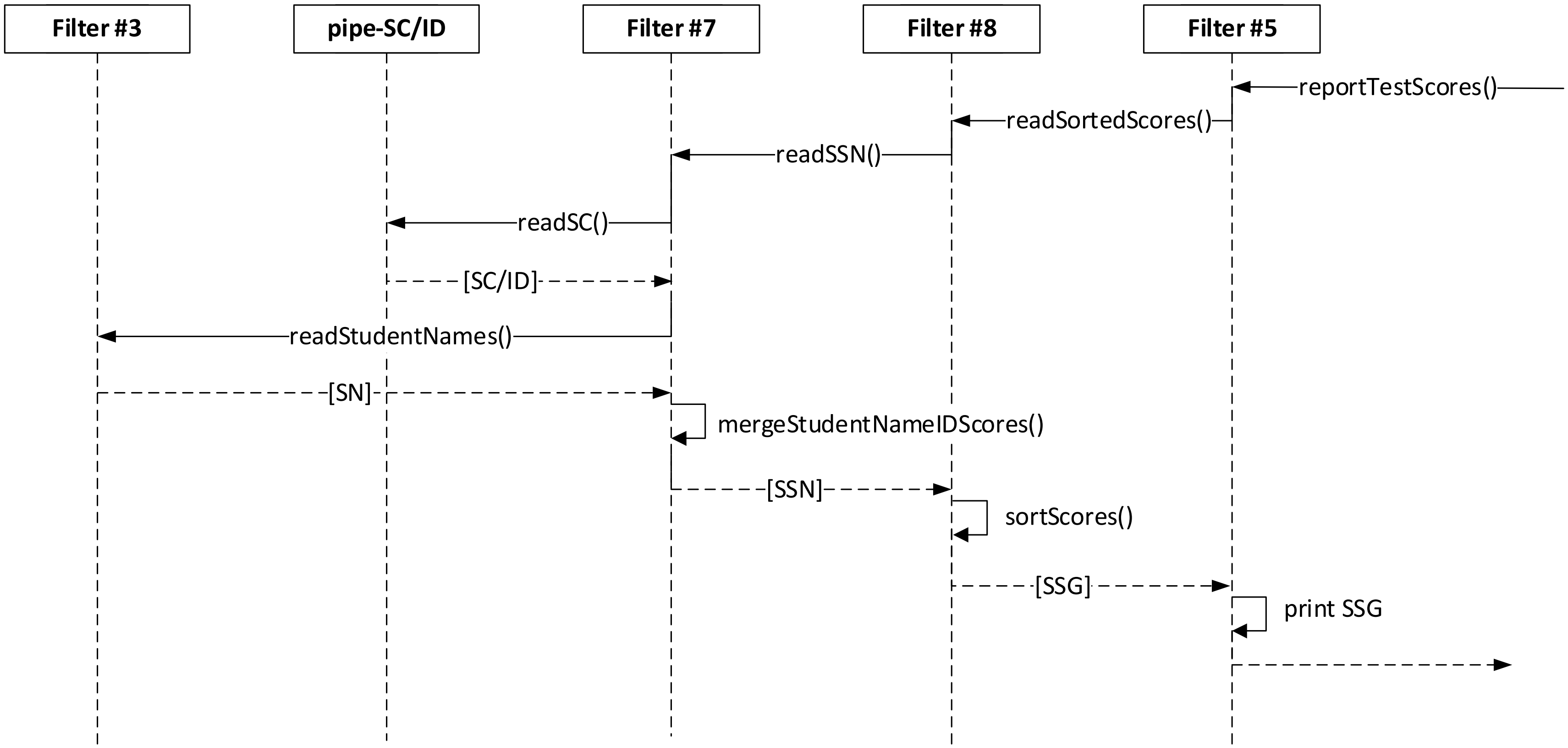
Class pipe-SC/ID
SC/ID // student's test scores together with student's ID

```
writeSC(SC/ID){
      store into SC/ID
}

readSC(){
       If SC/ID != NULL Then
           return SC
       Else
           return NULL
       End if
}
```

Sequence Diagram (Report test statics)



Cont. Sequence Diagram (Report test scores)



PROBLEM #2

There exist two library software systems S1 and S2 that maintain information about books in several libraries, i.e., a functional core of each library system is to keep track of the books in several libraries. Books may be checked-out or checked-in from the library and this should be reflected in the library system. Both library systems support the following operations:

Operations supported by the library system S1:

CheckingOut (user_id, book_id, library_id)	//a book is checked-out in a library by a user
CheckingIn (user_id, book_id, library_id)	//a book is checked-in in a library by user
ListBooks (user_id)	//list all books checked out by a user
IsBook (book_id)	//does a specified book exist in the libraries?

Operations supported by the library system S2:

Check_Out (book_id, library_id, user_id)	//a book is checked-out in a library by a user
Check_In (book_id, library_id, user_id)	//a book is checked-in in a library by user
List_Books (user_id)	//list all books checked out by a user
Is_Book (book_id)	//does a specified book exist in the libraries?

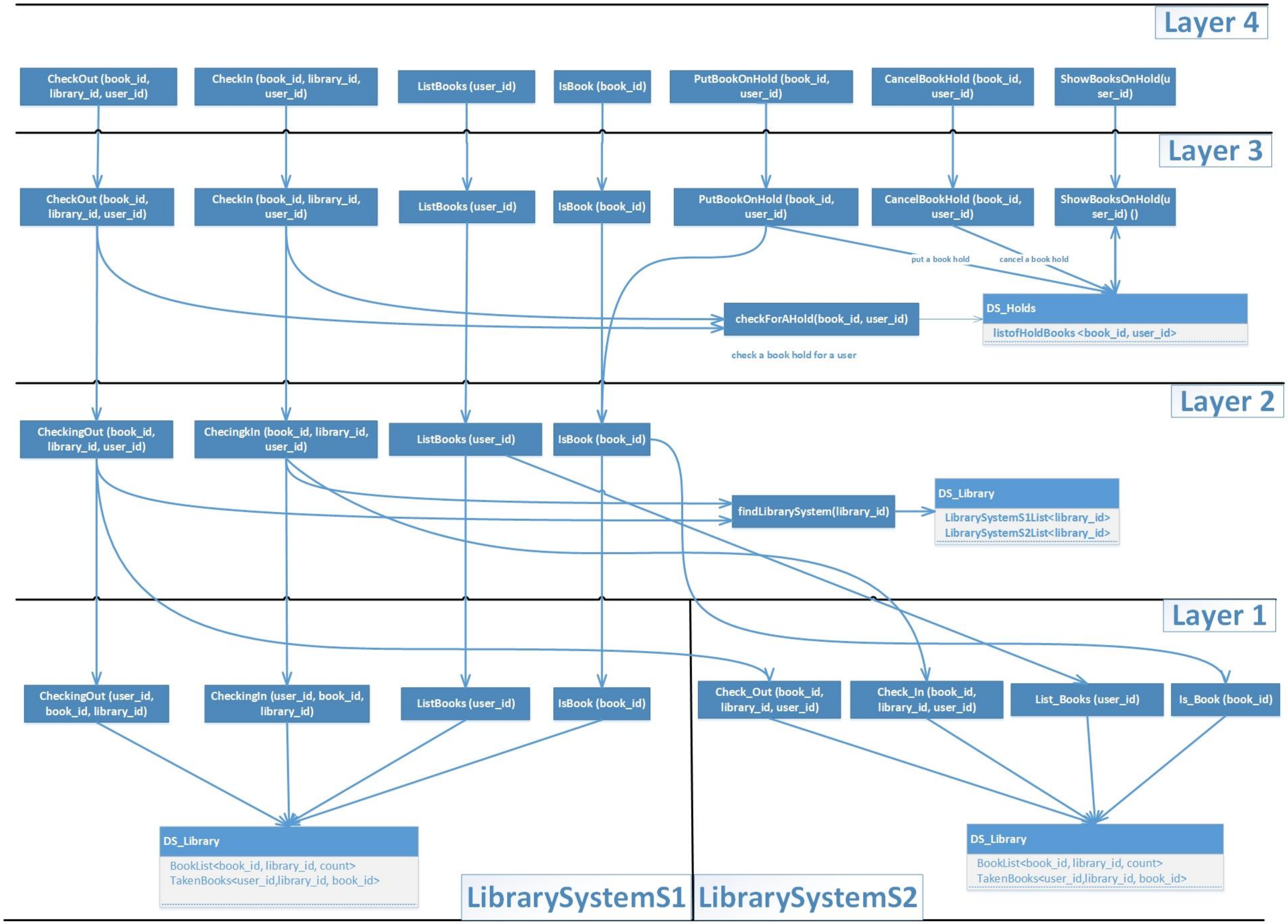
The goal is to combine both library systems and provide a uniform interface to perform operations on both existing library systems using the **Strict Layered architecture**. The following top layer interface should be provided:

CheckOut (book_id, library_id, user_id)	//a book is checked-out in a library by a user
CheckIn (book_id, library_id, user_id)	//a book is checked-in in a library by user
ListBooks (user_id)	//list all books checked out by a user
IsBook (book_id)	//does a specified book exist in the libraries?
PutBookOnHold (book_id, user_id)	//put a book on hold for a user
CancelBookHold (book_id, user_id)	//cancel a hold of a book for a user
ShowBooksOnHold(user_id)	//show all books on hold for a user

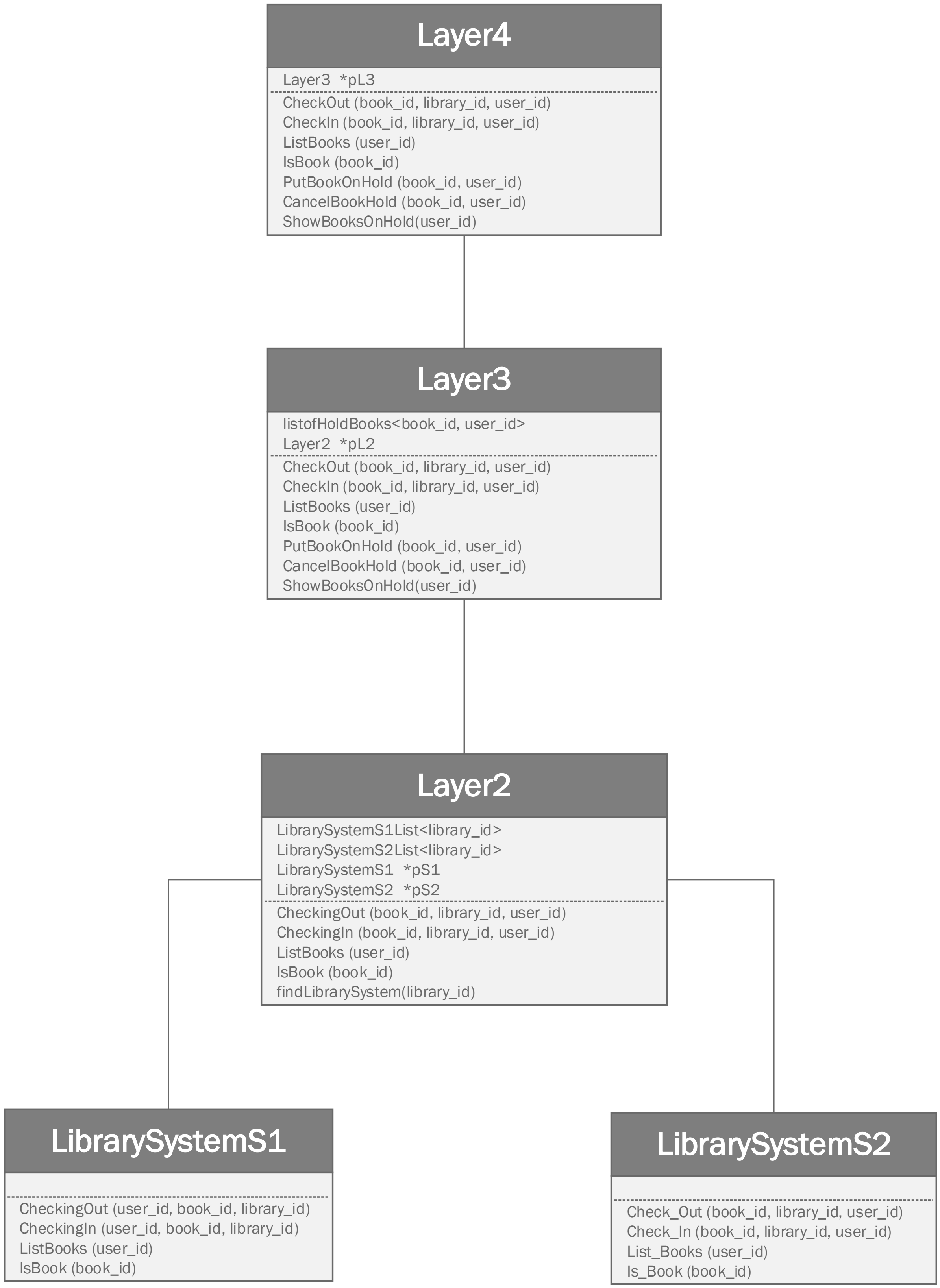
Notice that the top layer provides three additional functions (*PutBookOnHold()*, *CancelBookHold()*, and *ShowBooksOnHold()*) that are not provided by the existing library systems. These functions allow handling books that are on hold.

Major assumptions for the design:

1. Users/applications that use the top layer interface should have an impression that there exists only one library system.
2. The bottom layer is represented by both library systems (i.e., library systems S1 and S2).
3. Both library systems should not be modified.
4. Your design should contain at least **four** layers. For each layer identify operations provided by the layer.
5. Show call relationships between services of adjacent layers.
6. Each layer should be encapsulated in a class and represented by an object.
7. Provide a class diagram for the combined system. For each class list all operations supported by the class and major data structures. Briefly describe each operation in each class using **pseudo-code.**


Layer 4
CheckOut (book_id, library_id, user_id)
CheckIn (book_id, library_id, user_id)
ListBooks (user_id)
IsBook (book_id)
PutBookOnHold (book_id, user_id)
CancelBookHold (book_id, user_id)
ShowBooksOnHold(user_id)
Layer 3
CheckOut (book_id, library_id, user_id)
CheckIn (book_id, library_id, user_id)
ListBooks (user_id)
IsBook (book_id)
PutBookOnHold (book_id, user_id)
CancelBookHold (book_id, user_id)
ShowBooksOnHold(user_id) ()
put a book hold
cancel a book hold
checkForAHold(book_id, user_id)
check a book hold for a user
DS_Holds
listofHoldBooks <book_id, user_id>
Layer 2
CheckingOut (book_id, library_id, user_id)
ChecingkIn (book_id, library_id, user_id)
ListBooks (user_id)
IsBook (book_id)
findLibrarySystem(library_id)
DS_Library
LibrarySystemS1List<library_id>
LibrarySystemS2List<library_id>
Layer 1
CheckingOut (user_id, book_id, library_id)
CheckingIn (user_id, book_id, library_id)
ListBooks (user_id)
IsBook (book_id)
Check_Out (book_id, library_id, user_id)
Check_In (book_id, library_id, user_id)
List_Books (user_id)
Is_Book (book_id)
DS_Library
BookList<book_id, library_id, count>
TakenBooks<user_id,library_id, book_id>
DS_Library
BookList<book_id, library_id, count>
TakenBooks<user_id,library_id, book_id>
LibrarySystemS1
LibrarySystemS2

Class Diagram



Class Layer4:

Layer3 *pL3

```
CheckOut(book_id, library_id, user_id){
        pL3 -> CheckOut(book_id, library_id, user_id)
}

CheckIn(book_id, library_id, user_id){
        pL3 -> CheckIn(book_id, library_id, user_id)
}

ListBooks(user_id){
        return pL3 -> ListBooks(user_id)
}

IsBook(book_id){
        return pL3 -> IsBook(book_id)
}

PutBookOnHold(book_id, user_id){
        pL3 -> PutBookOnHold(book_id, user_id)
}

CancelBookHold(book_id, user_id){
        pL3 -> CancelBookHold(book_id, user_id)
}

ShowBooksOnHold(user_id){
        pL3 -> ShowBooksOnHold(user_id)
}
```

Class Layer3:

listofHoldBook
Layer2 *pL2

```
CheckOut(book_id, library_id, user_id){
        If checkForAHold(book_id, user_id) == false Then
           pL2 -> CheckingOut(book_id, library_id, user_id)
        Else
```

```
            reject the checkout request because of the hold
        End If
}

CheckIn(book_id, library_id, user_id){
        pL2 -> CheckingIn(book_id, library_id, user_id)
}

ListBooks(user_id){
        return pL2 -> ListBooks(user_id)
}

IsBook(book_id){
        return pL2 -> IsBook(book_id)
}

PutBookOnHold(book_id, user_id){
        If ( pL2 -> IsBook (book_id) == true )  Then
              insert <book_id, user_id> pair into listOfHoldBooks
        End If
}

CancelBookHold(book_id, user_id){
        delete <book_id, user_id> pair from listOfHoldBooks
}

ShowBooksOnHold(user_id){
        for each <book_id, user_id> pair in listOfHoldBooks
                If user_id matches in the pair Then
                        display the pair's book_id
                End If
}

checkForAHold(book_id, user_id){
        If <book_id, user_id> pair exists in listOfHoldBooks Then
                return true
        Else
                return false
        End If
}
```

Class Layer2:

LibrarySystemS1List<library_id>
LibrarySystemS2List<library_id>
LibrarySystemS1 *pS1
LibrarySystemS2 *pS2

```
CheckOut(book_id, library_id, user_id){
    If findLibrarySystem(library_id) == 1 Then
        pS1-> CheckingOut (user_id, book_id, library_id)
    Else findLibrarySystem(library_id) == 2 Then
        pS2-> Check_Out (book_id, library_id, user_id)
    End If
}

CheckIn(book_id, library_id, user_id){
    If findLibrarySystem(library_id) == 1 Then
        pS1-> CheckingIn (user_id, book_id, library_id)
    Else findLibrarySystem(library_id) == 2 Then
        pS2-> Check_In (book_id, library_id, user_id)
    End If
}

ListBooks(user_id){
    bookList1 = pS1 -> ListBooks (user_id)
    bookList2 = pS2 -> List_Books (user_id)
    merge bookList1 and bookList2 into bookList
    return bookList
}

IsBook(book_id){
      If ( pS1 -> IsBook (book_id) == true )  or  (pS2 -> Is_Book (book_id)   == true)   Then
          return true  // yes, a specified book exists in the libraries
      Else
          return false // no, a specified book does exist in the libraries
      End If
}
```

```
findLibrarySystem(library_id){
        If library_id is in LibrarySystemS1List Then
           return 1
        Else If  library_id  is in LibrarySystemS2List Then
           return 2
        Else
           library is not found error
        End If
}
```

PROBLEM #3

Suppose that we would like to use a fault-tolerant architecture for the *RemoveDuplicates* component that removes duplicates from a list of integers within *low-high* range. The *unique()* operation of this component accepts as an input integer parameters *n*, *low*, *high* and an integer array *L*. The component removes duplicates which values are greater or equal to *low* but smaller or equal to *high*. The output parameters are (1) an integer array *SL* that contains the list of integers from list *L* without duplicates within *low-high* range and (2) an integer *m* that contains the number of elements in list *SL*. An interface of the *unique()* operation is as follows:

void unique (**in** int n, int low, int high, int L[]; **out** int SL[], int m)

L is an array of integers,
n is the number of elements in list *L*,
low is the lower bound for removing duplicates,
high is the upper bound for removing duplicates,
SL is an array of unique integers from list *L*,
m is the number of unique elements in list *SL*
Notice: *L* and *n* are inputs to the *unique()* operation. *SL* and *m* are output parameters of the *unique()* operation.

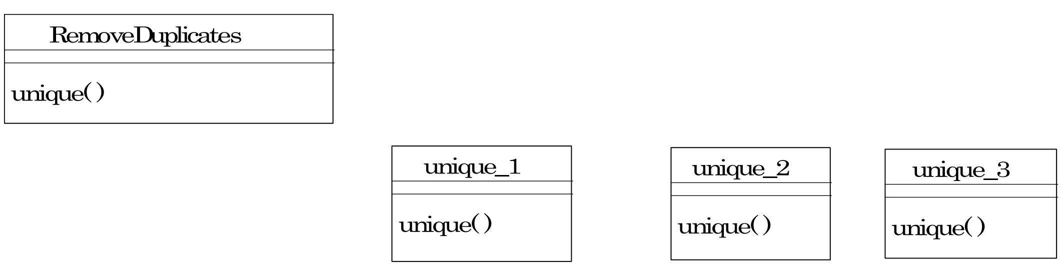
For example, for the following input:

n=8, *low*=2, *high*=6, *L*=(1, 7, 1, 2, 5, 2, 7, 5)

the *unique*() operation returns the following output parameters:

m=6, *SL*=(1, 7, 1, 2, 5, 7)

Suppose that three versions of the *RemoveDuplicates* component have been implemented using different algorithms. Different versions are represented by classes: *unique_1*, *unique_2, unique_3*.



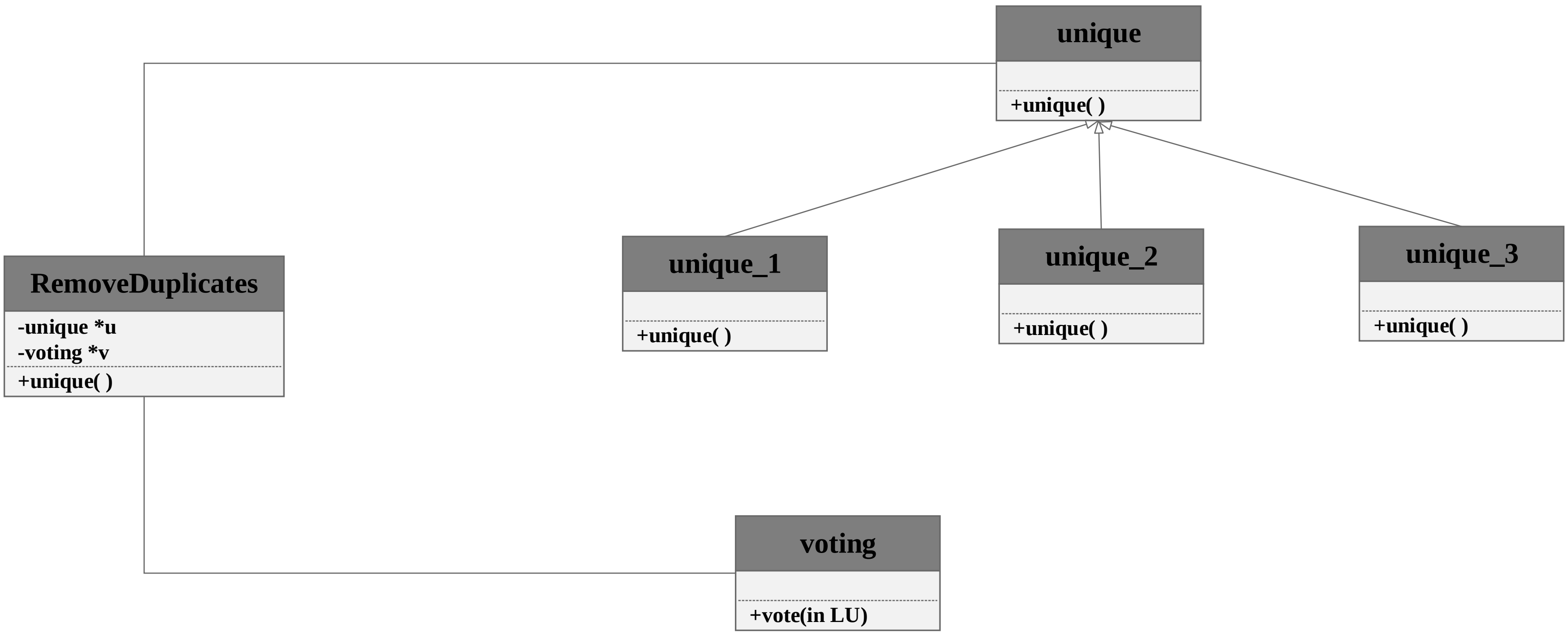
Provide two designs for the *RemoveDuplicates* component using the following types of fault-tolerant software architectures:

1. N-version architecture
2. Recovery-Block architecture

For each design provide a class diagram. For each class identify operations supported by the class and its attributes. Specify in detail each operation using pseudo-code (you do not need to specify operations *unique()* of the *unique_i* classes; only new operations need to be specified). For each design provide a sequence diagram representing a typical execution of the *RemoveDuplicates* component.

1. N-version architecture

Class Diagram



Pseudo code

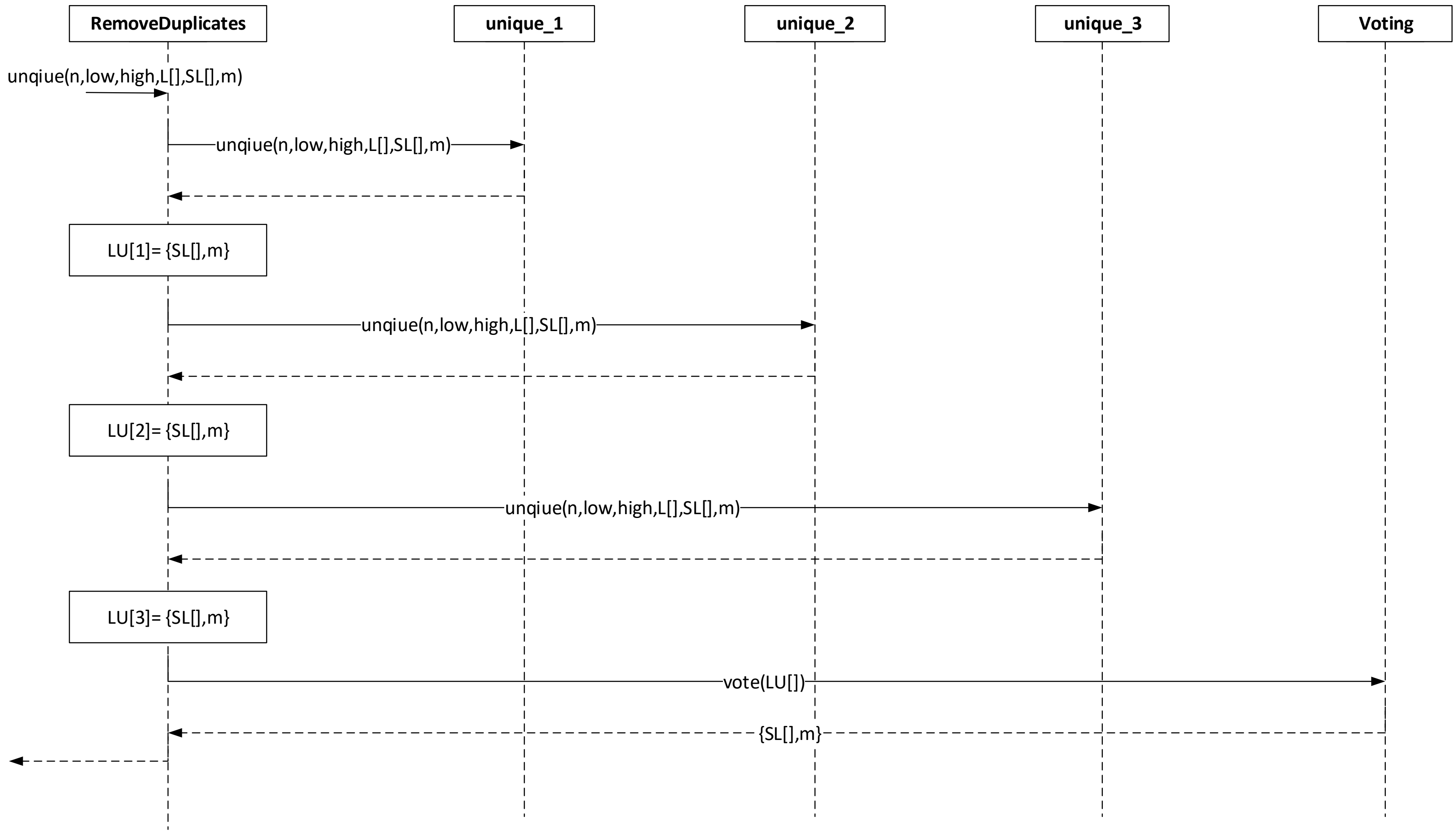
Class RemoveDuplicates:

```
unique *u        // a pointer to the unique object
voting *v        // a pointer to the voting object


void unique(in int n, int low, int high, int L[]; out int SL[], int m){
    LU[] is a { int[], int } list    // {integer list, integer size of the list}
    unique u[]
    u[1] = new unique_1()
    u[2] = new unique_2()
    u[3] = new unique_3()
    For i = 1 to 3
        u[i]->unique(n, low, high, L[], SL[], int m)
        LU[i] = {SL[], m}
    End For
    {SL[], m} = v->vote(LU)
}
```

Class voting:

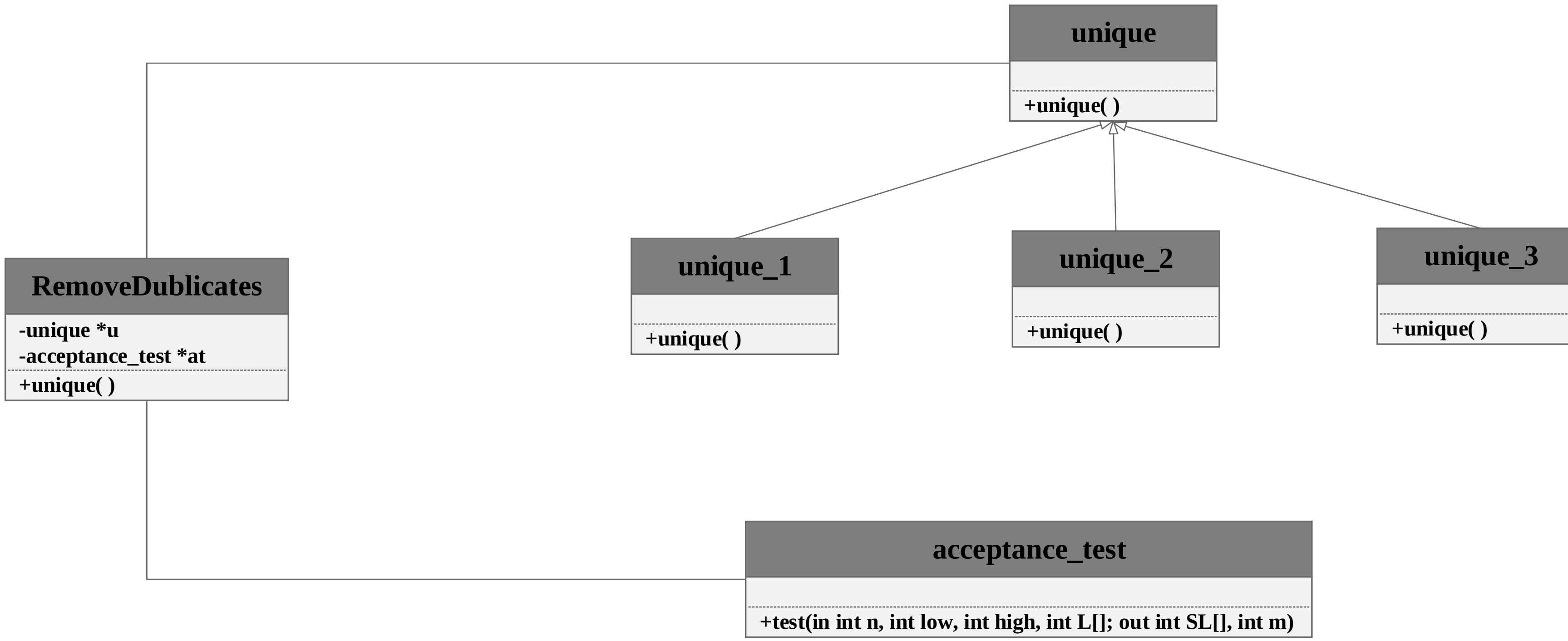
```
{int[], int} vote(in: LU){
    If LU[1] == LU[2] Then
        return LU[1]
    Else If LU[2] == LU[3] Then
        return LU[2]
    Else If LU[1] == LU[3] Then
        return LU[3]
    ENDIF
    r = Random(1,3) // generate random number between 1 to 3
    return LU[r]  //randomly select one from LU and return
}
```

Sequence Diagram



2. Recovery-Block architecture

<u>**Class Diagram**</u>



<u>Pseudo code</u>

Class RemoveDuplicates:

```
unique *u                    // a pointer to the unique object
acceptance_test *at          // a pointer to the acceptance_test object


void unique(in int n, int low, int high, int L[]; out int SL[], int m){
        LU[] is a { int[], int } list      // {integer list, integer size of the list}
        unique u[]


        u[1] = new unique_1()
        u[1]-> unique(n, low, high, L[], SL[], int m)
        LU[1] = {SL[], m}
        testResult = at->test(n, low, high, L[], SL[], int m)
        If testResult == true Then
            exit;
        End If
        u[2] = new unique_2()
        u[2]-> unique(n, low, high, L[], SL[], int m)
        LU[2] = {SL[], m}
        testResult = at->test(n, low, high, L[], SL[], int m)
        If testResult == true Then
```

exit;

End If

u[3] = new unique_3()

u[3]-> unique(n, low, high, L[], SL[], int m)

LU[3] = {SL[], m}

testResult = at->test(n, low, high, L[], SL[], int m)

If testResult == true **Then**

exit;

End If

// if all tests are false

r = **Random**(1,3) // generate random number between 1 to 3

{SL[], m} = LU[r] //randomly select one from LU

}

Class acceptance_test:

```
boolean test (in int n, int low, int high, int L[]; out int SL[], int m){
    If m > n Then  //if the size of the output list is greater than the size of the original list
        return false;
    End If
    // a hash table to detect duplicates in the SL[]
    bool UniqueTable[m] = {false}  //if UniqueTable[x] is true, then x is in the list SL[]
    For i = 0 to m-1
        If SL[i] >= low and SL[i] <= high Then      //if SL[i] is within low-high range
            If UniqueTable[ SL[i] ]  ==  true Then   //if  SL[i] has already appeared in the list
                return false;
            Else
                UniqueTable[ SL[i] ]  = true;  // set SL[i] in the list
            End If
        End If
    End For
    return true;
}
```

Sequence Diagram