Com1101 Exam 2 – Winter 2003

Name: _____

Student Id (last 4 digits) : _____

• Write down the answers in the space provided. Use what you know from the lectures, labs, and the assigned readings.
• Please remember that the phrase "develop a function" means to design a function according to one of the recipes. You are *not* required to provide a template unless the problem specifically asks for one.
• You may obtain a maximum of 60 points.

Good luck.

Problem	Points
1(10)	
2(15)	
3(20)	
4(10)	
5(10)	
Total (out of 55)	
Base	55

10 Points

1 Problem

Design a class of Books with title, author, and price in dollars.

```
class Book {
    /*-----------------------------------------------------------
     The member data
    -----------------------------------------------------------*/
    /* Title of the book  */
    String title;

    /* Author of the book */
    String author;

    /* Price of the book  */
    int price;

    /*-----------------------------------------------------------
     The constructor
    -----------------------------------------------------------*/
    Book(String aTitle, String anAuthor, int aPrice){
        this.title  = aTitle;
        this.author = anAuthor;
        this.price  = aPrice;
    }
/*************************************************************/
/*-----------------------------------------------------------
     Define a Book object and print the member data values.
    -----------------------------------------------------------*/
    void TestBook(){
        testHeader("Book class");

        println("\nDefine Book objects, print member data");
        println(new Book("Bard", "Othello", 12));
        println(new Book("Bard", "Macbeth", 7));
        println(new Book("Wolf", "Room",    8));
        println(new Book("Rilke", "Poet",   15));
    }
```

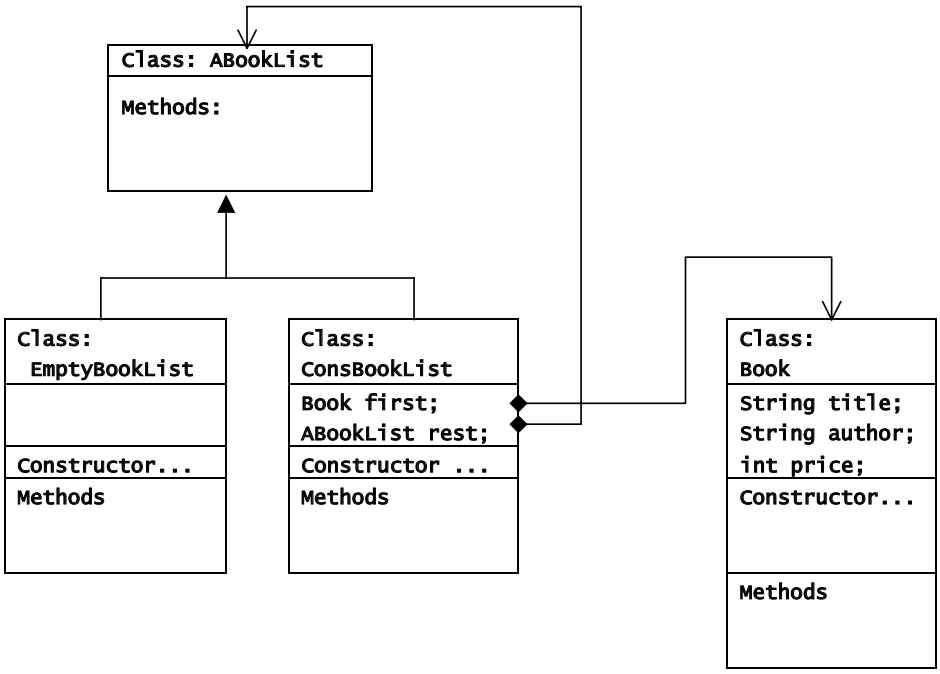
Design a set of classes that represent a list of Books.

```
/* AListBooks.java */
    abstract class AListBooks { }

/* EmptyListBooks.java */
    class EmptyListBooks extends AListBooks{ }

/* ConsListBooks.java */ class ConsListBooks extends AListBooks{
    /*-------------------------------------------------------
     Member data:
     -------------------------------------------------------*/
    Book       first;
    AListBooks rest;

    /*-------------------------------------------------------
     The constructor
    -------------------------------------------------------*/
    ConsListBooks(Book aFirst, AListBooks aRest){
        this.first = aFirst;
        this.rest  = aRest;
    }
/*********************************************************/
    /*-------------------------------------------------------
     Test the list of Books.
    -------------------------------------------------------*/
    void TestListBooks(){
        testHeader("List of Books classes");

        Book othello = new Book("Bard", "Othello", 12);
        Book macbeth = new Book("Bard", "Macbeth", 7);
        Book room    = new Book("Wolf", "Room",    8);
        Book poet    = new Book("Rilke", "Poet",   15);

        AListBooks empty = new EmptyListBooks();

        AListBooks shortList = new ConsListBooks(othello,
                               new ConsListBooks(macbeth,
                               empty));
```

```
    AListBooks longList  = new ConsListBooks(othello,
                           new ConsListBooks(macbeth,
                           new ConsListBooks(room,
                           new ConsListBooks(poet,
                           empty))));

    actual  (empty);
    actual  (shortList);
    actual  (longList);
}
```

Draw the UML diagram for this class hierarchy.

A List of Books



Design the method `cheapBooks` for the list of Books, which produces a new list of those books that cost less than $10.

```
    /*-------------------------------------------------------
     Method in class AListBooks
    -------------------------------------------------------*/
    abstract AListBooks cheapBooks();

/*************************************************************/
    /*-------------------------------------------------------
     Method in class EmptyListBooks
    -------------------------------------------------------*/
    AListBooks cheapBooks(){
        return this;
    }
```

```
/*************************************************************/
    /*-----------------------------------------------------
     Method in class ConsListBooks
    -----------------------------------------------------*/
    AListBooks cheapBooks(){

        if (this.first.price < 10)
            return new ConsListBooks(this.first,
                                     this.rest.cheapBooks());
        else
            return this.rest.cheapBooks();
    }
/*************************************************************/
    /*-----------------------------------------------------
     Test the method cheapBooks.
    -----------------------------------------------------*/
    void TestCheapBooks(){
        testHeader("cheapBooks");

        Book othello = new Book("Bard", "Othello", 12);
        Book macbeth = new Book("Bard", "Macbeth", 7);
        Book room    = new Book("Wolf", "Room",    8);
        Book poet    = new Book("Rilke", "Poet",   15);

        AListBooks empty = new EmptyListBooks();
        AListBooks shortList = new ConsListBooks(othello,
                               new ConsListBooks(poet,
                               empty));
        AListBooks longList  = new ConsListBooks(othello,
                               new ConsListBooks(macbeth,
                               new ConsListBooks(room,
                               new ConsListBooks(poet,
                               empty))));
        expected(empty);
        actual  (shortList.cheapBooks());

        expected(new ConsListBooks(macbeth,
                 new ConsListBooks(room, empty)));
        actual  (longList.cheapBooks());
    }
```

15 Points

2 Problem

Design a set of classes that represents a list of lists of characters. You do not need to develop any methods within these classes, unless you are specifically asked to do so. Do include the headers for the necessary constructors.

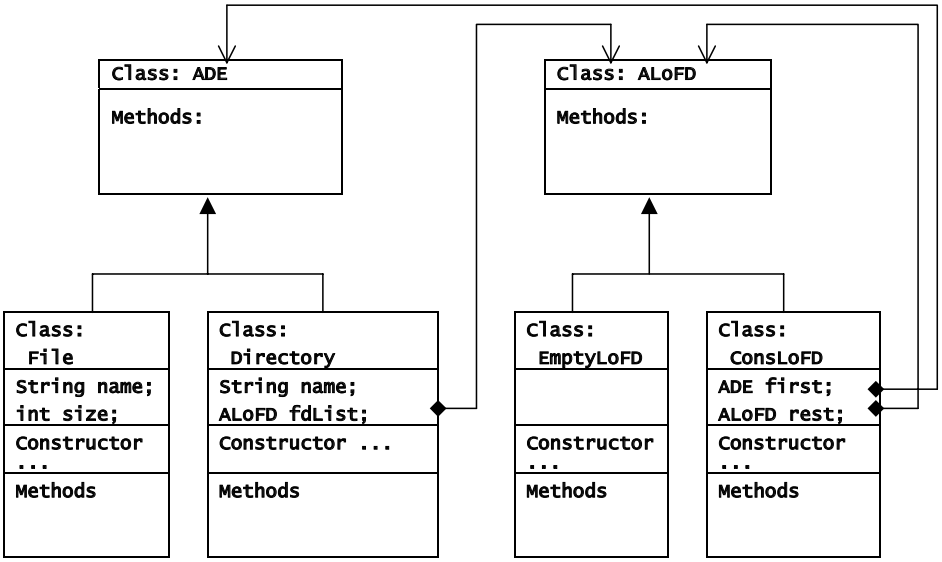
```
/*-------------------------------------------------------------
AListChars.java ------------------------------------------*/
abstract class AListChars {

    /* count the number of items in this list*/
    abstract int count();
}

/*-------------------------------------------------------------
EmptyListChars.java --------------------------------------*/
class EmptyListChars extends AListChars{

    int count(){
        return 0; }
}

/*-------------------------------------------------------------
ConsListChars.java ---------------------------------------*/
class ConsListChars extends AListChars{
    /*-----------------------------------------------------
     Member data:
     -----------------------------------------------------*/
    char       first;
    AListChars rest;

    /*-----------------------------------------------------
     The constructor
    -------------------------------------------------------*/
    ConsListChars(char aFirst, AListChars aRest){
        this.first = aFirst;
        this.rest  = aRest;
    }
```

```
    /*-------------------------------------------------------
     Method templates:
    -------------------------------------------------------*/
    /*   ...this.first...this.rest...this.rest.count()...   */

    /*-------------------------------------------------------
     Methods:
    -------------------------------------------------------*/
    /* count the number of items in this list */
    int count(){
        return 1 + this.rest.count();
    }

/*-------------------------------------------------------------
ALoLChars.java ------------------------------------------------*/
abstract class ALoLChars {

    /* count the number of items in each list of this list*/
    abstract AListIntegers count();
}

/*-------------------------------------------------------------
EmptyLoLChars.java --------------------------------------------*/
class EmptyLoLChars extends ALoLChars{

    AListIntegers count(){
        return new EmptyListIntegers();
    }
}

/*-------------------------------------------------------------
ConsLoLChars.java ---------------------------------------------*/
class ConsLoLChars extends ALoLChars{

    /*-------------------------------------------------------
     Member data:
     -------------------------------------------------------*/
    AListChars first;
    ALoLChars  rest;
    /*-------------------------------------------------------
```

```
 The constructor
-------------------------------------------------------*/
ConsLoLChars(AListChars aFirst, ALoLChars aRest){
    this.first = aFirst;
    this.rest  = aRest;
}
/*-------------------------------------------------------
 Method templates:
-------------------------------------------------------*/
/*  ...this.first...this.rest...
    ...int this.first.count()...
    ...AListIntegers this.rest.count()...              */

/*-------------------------------------------------------
 Methods:
-------------------------------------------------------*/
AListIntegers count(){
    return new ConsListIntegers(this.first.count(),
                                this.rest.count());
}
```

Draw the UML diagram for this class hierarchy.

A List of Files and Directories



Define the method `itemCount` for this list of lists, which produces a new list of integers, each representing the number of items in the corresponding list. That means, the first integer in the new list tells us how many items were in the first sub-list of characters, etc.

```
/*-----------------------------------------------------
 Define a list of lists of characters
 and print the member data values.
-----------------------------------------------------*/
void TestLoLChars(){

    println("\nDefine list of lists of chars,
                    print member data values");

    AListChars emptyChars = new EmptyListChars();

    AListChars fewChars =   new ConsListChars('C',
                            new ConsListChars('A',
```

```
                        new ConsListChars('T',
                        new EmptyListChars(  ) )));

AListChars manyChars =  new ConsListChars('A',
                        new ConsListChars('B',
                        new ConsListChars('C',
                        new ConsListChars('D',
                        new ConsListChars('E',
                        new ConsListChars('F',
                        new ConsListChars('G',
                        new EmptyListChars(  ) )))))));

testHeader("lists of characters");

expected(0);
actual(emptyChars.count());

expected(3);
actual(fewChars.count());

expected(7);
actual(manyChars.count());

ALoLChars emptyLoL = new EmptyLoLChars();

ALoLChars shortLoL = new ConsLoLChars(fewChars,
                     new EmptyLoLChars());

ALoLChars longLoL  = new ConsLoLChars(fewChars,
                     new ConsLoLChars(emptyChars,
                     new ConsLoLChars(manyChars,
                     new EmptyLoLChars())));

testHeader("lists of lists of characters");

println(emptyLoL);
println(shortLoL);
println(longLoL);

testHeader("list of counts");
```

```
    expected(new EmptyListIntegers());
    actual(emptyLoL.count());

    expected(new ConsListIntegers(3,
                                  new EmptyListIntegers()));
    actual(shortLoL.count());

    expected(new ConsListIntegers(3,
             new ConsListIntegers(0,
             new ConsListIntegers(7,
             new EmptyListIntegers()))));
    actual(longLoL.count());
}
```

20 Points

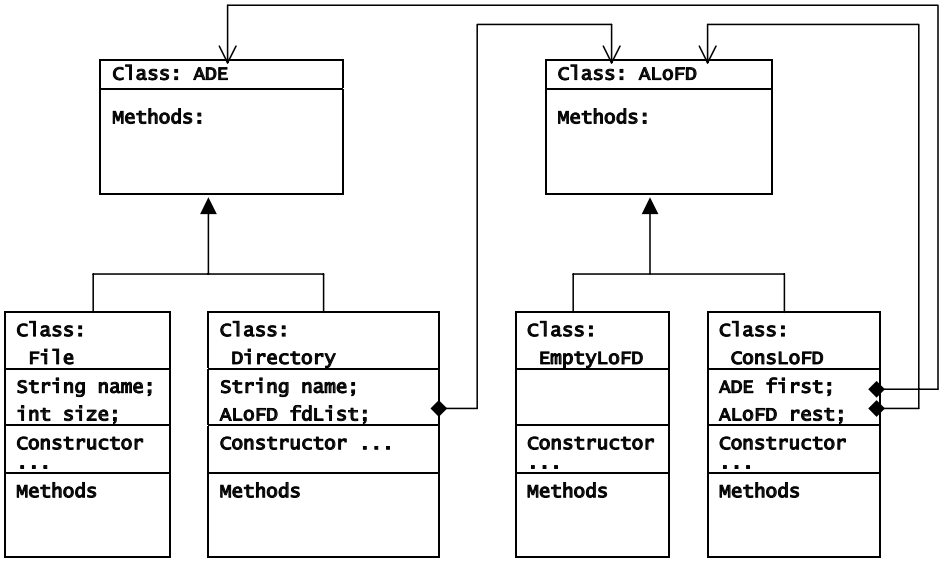
3 Problem

Given the following data definitions for **A List of Files and Directories** together with the skeleton of the UML for this class hierarchy:
`File` is a structure with
    `String` name
    `int` size
`Directory` is a structure with
    `String` name
    `ALoFD` fdList
ADirectoryEntry (`ADE`)is
    either a `File`
    or a `Directory`
A List of Files and Directories (`ALoFD`) is
    either `EmptyLoFD`
    or `ConsLoFD`
`EmptyLoFD` has no attributes/fields
`ConsLoFD` is a structure with
    `ADE` first
    `ALoFD` rest

perform the following tasks:

- Add the missing arrows to the UML diagram.

A List of Files and Directories



- Design the template for each class as needed, to develop the method `int size();`, which computes the total size of all files in this list of files and directories.

```
/* in class File ***************************************/
    /*-------------------------------------------------
     Method template:
     ...this.name...this.size...this.size() */

/* in class Directory ***********************************/
    /*-------------------------------------------------
     Method template:
     ...this.name...this.fdList...this.fdList.size()...      */

/* in class ConsLoFD ************************************/
    /*-------------------------------------------------
     Method template:
    /*  ...this.first... this.first.size()...
        ...this.rest...this.rest.size()...                  */
```

- Implement the `Comparable interface` for the `class ADE`.

```
/*-----------------------------------------------------------
ADE.java -----------------------------------------------------*/
abstract class ADE implements Comparable{

    /* compute the size of all objects in this list */
    abstract public int size();

    /* compare two ADE objects */
    public int compareTo(Object obj){
        return (this.size() - ((ADE)obj).size());
    }
}

/*************************************************************/
/* Test suite ************************************************/
        File pics  = new File("Pictures", 500);
        File songs = new File("Songs", 750);
        File docs  = new File("Documents", 300);
        File progs = new File("Programs", 300);

        ALoFD emptyList = new EmptyLoFD();

        ALoFD fewFiles =   new ConsLoFD(pics,
                               new ConsLoFD(songs,
                               new ConsLoFD(progs,
                               emptyList)));

        ALoFD manyFiles =  new ConsLoFD(docs, fewFiles);

/* additional examples ***************************************/
        Directory dir1 = new Directory("Dir1", fewFiles);
        Directory dir2 = new Directory("Dir2", manyFiles);
        ALoFD bigOne = new ConsLoFD(dir2, fewFiles);

        /* test compareTo() method */

        expected("<0");
        actual(pics.compareTo(songs));
```

```
expected(">0");
actual(dir1.compareTo(docs));

expected("<0");
actual(dir1.compareTo(dir2));

expected(">0");
actual(pics.compareTo(progs));

expected("=0");
actual(docs.compareTo(progs));
```

- Design the method `sort()`, which sorts the directory entries in the `ALoFD` in ascending order. Use templates to develop this method. You will need an additional helper method. Follow the design recipe for the helper method as well.

```
/* in class ALoFD ***************************************/
    /*-------------------------------------------------
    /* sort the list by the size */
    abstract ALoFD sort();

    /* insert the given ADE into the sorted list
    of files and directories */
    abstract ALoFD insert(ADE someADE);

/* in class EmptyLoFD ***********************************/
    /*-------------------------------------------------
    ALoFD sort(){
        return this;
    }

    ALoFD insert(ADE someADE){
        return new ConsLoFD(someADE, this);
    }

/* in class ConsLoFD ************************************/
    /*-------------------------------------------------
     Method template:
    /*  ...this.first...this.first.size()...
        ...this.first.compareTo(someADE)...
        ...this.rest...this.rest.size()...
        ...this.rest.insert(someADE)
        ...this.rest.sort()...                  */

    /*-------------------------------------------------
    ALoFD sort(){
        return this.rest.sort().insert(this.first);
    }

    /* insert the given ADE into the sorted list
```

```
of files and directories */
ALoFD insert(ADE someADE){
    if (this.first.compareTo(someADE) < 0)
      return new ConsLoFD(this.first,
                          this.rest.sort().insert(someADE));
    else
      return new ConsLoFD(someADE, this);
}
```

```
/*********************************************************
* Test suite for Problem 3 */

/*-------------------------------------------------------
 Define a list of file objects and
 print the member data values.
-------------------------------------------------------*/
void TestFiles(){
    println("\nDefine file objects, print member data");

    println(new File("Pictures", 500));
    println(new File("Songs", 750));
    println(new File("Documents", 200));
    println(new File("Programs", 300));
}

/*-------------------------------------------------------
 Define a list of files and directories
 and print the member data values.
-------------------------------------------------------*/
void TestALoFD(){

    File pics  = new File("Pictures", 500);
    File songs = new File("Songs", 750);
    File docs  = new File("Documents", 200);
    File progs = new File("Programs", 300);

    print("\nDefine list of files and directories, )
    println("print member data values");

    ALoFD emptyList = new EmptyLoFD();

    ALoFD fewFiles =   new ConsLoFD(pics,
                           new ConsLoFD(songs,
                           new ConsLoFD(progs,
                           emptyList)));

    ALoFD manyFiles =  new ConsLoFD(docs, fewFiles);

    Directory dir1 = new Directory("Dir1", fewFiles);
```

```
    Directory dir2 = new Directory("Dir2", manyFiles);
    ALoFD bigOne = new ConsLoFD(dir2, fewFiles);

    /* test the directories construction */
    testHeader("directories");

    println(dir1);
    println(dir2);

    /* test the list of files and directories construction */
    testHeader("lists of files and directories");

    println(emptyList);
    println(fewFiles);
    println(manyFiles);
    println(bigOne);

    /* test the size() method */
    testHeader("size()");

    expected(0);
    actual(emptyList.size());

    expected(1550);
    actual(fewFiles.size());

    expected(1750);
    actual(manyFiles.size());

    /* test the sort() method */
    testHeader("sort()");

    actual(manyFiles.sort());
    actual(bigOne.sort());
}
```

10 Points

4 Problem

Given iterator interface `IRange`, a structure `Struct` of `Items`. `Item` implements `IMeasurable` (with method '`int size();`'). In addition, there is an iterator `StructRange` for the structure `Struct`, which implements `IRange`:

- Write the test (external) method `totalSize`, which computes the total size of all items in the `Struct`. Use the `for` loop.

```
void TestTotalSize(){

    println("\nTest totalSize() method:");

    Item i1 = new Item("Pictures");
    Item i2 = new Item("Songs");
    Item i3 = new Item("Documents");
    Item i4 = new Item("Programs");
    Struct s = new Struct(new Item[]{i1, i2, i3, i4});

    expected(30);
    actual(totalSize(s));
}

/* compute total size of the items in the given structure */
int totalSize(Struct myStruct){

    int total = 0;
    for (IRange it = new StructRange(myStruct);
         it.hasMore();
         it.next()){

        total = total + ((Item)it.current()).size();
    }
    return total;
}
```

10 Points

5 Problem

Given iterator interface `IRange`, a structure `ManyThings` of `Thing`, which implements `ISearchable`

(with method `boolean sameThing(Thing someThing);`

and its iterator `ThingRange`, which implements `IRange`:

Write the test (external) method `findThing`, which will determine whether a given object appears in this structure. Use the `for` loop!

```
void TestfindThing(){
    println("\nTest findThing() method:");
    Thing i1 = new Thing("Pictures");
    Thing i2 = new Thing("Songs");
    Thing i3 = new Thing("Documents");
    Thing i4 = new Thing("Programs");
    ManyThings s = new ManyThings(new Thing[]{i1, i2, i3, i4});

    expected(true);
    actual(findThing(s, i2));

    expected(false);
    actual(findThing(s, new Thing("Videos")));
}

/* determine whether a given Thing
   appears in the given structure */
boolean findThing(ManyThings myThings, Thing someThing){

    for (IRange it = new ThingRange(myThings);
         it.hasMore();
         it.next()){

        if (someThing.sameThing( ((Thing)it.current()) ))
            return true;
    }
```

```
    return false;
}
```

Addenda

Problem 2

Read the whole problem first. The first part does not ask you to develop methods, but the remaining parts do. Make sure you leave space for that work, so you do not have to rewrite things.

Problem 3

Again, read the whole problems and plan your work.
Use the examples below, but **add three more examples of your own**. You may use the objects defined here as parts of your examples.

```
File pics  = new File("Pictures", 500);
File songs = new File("Songs", 750);
File docs  = new File("Documents", 200);
File progs = new File("Programs", 300);

ALoFD emptyList = new EmptyLoFD();

ALoFD fewFiles =   new ConsLoFD(pics,
                        new ConsLoFD(songs,
                        new ConsLoFD(progs,
                        emptyList)));

ALoFD manyFiles =  new ConsLoFD(docs, fewFiles);
```

Problems 4 and 5

- You **do not need to draw UML diagrams** in either of these problems.
- **Only one of these two problems is required** - you may choose which one. The other problem will be counted as extra credit.
- You may think of `Struct` as a `List` or `Array` structure of `Item`.
- You may think of `ManyThings` as a `List` or `Array` structure of `Thing`.