Lab 10

Demo Points (50 Marks)

Stories 14 – 24 (5 Marks each)

- JUnit Test (3 Marks)
- LibraryInOut method (2 Marks)

This Week

This week you will refactor your input/output class so that only the streams are in the class. The read and print functions for the objects are contained in the objects. The objects retrieve the streams from the I/O class. This is a better design because the LibraryInputOutput is less coupled to the rest of the classes. The I/O class does not need to know the structure of the classes that will be printed. Instead, each class will print itself using the streams managed by the I/O class.

The refactored classes will follow the UML diagram below. Notice that the print and read functions have been moved to the Friend, Item and Loan classes.

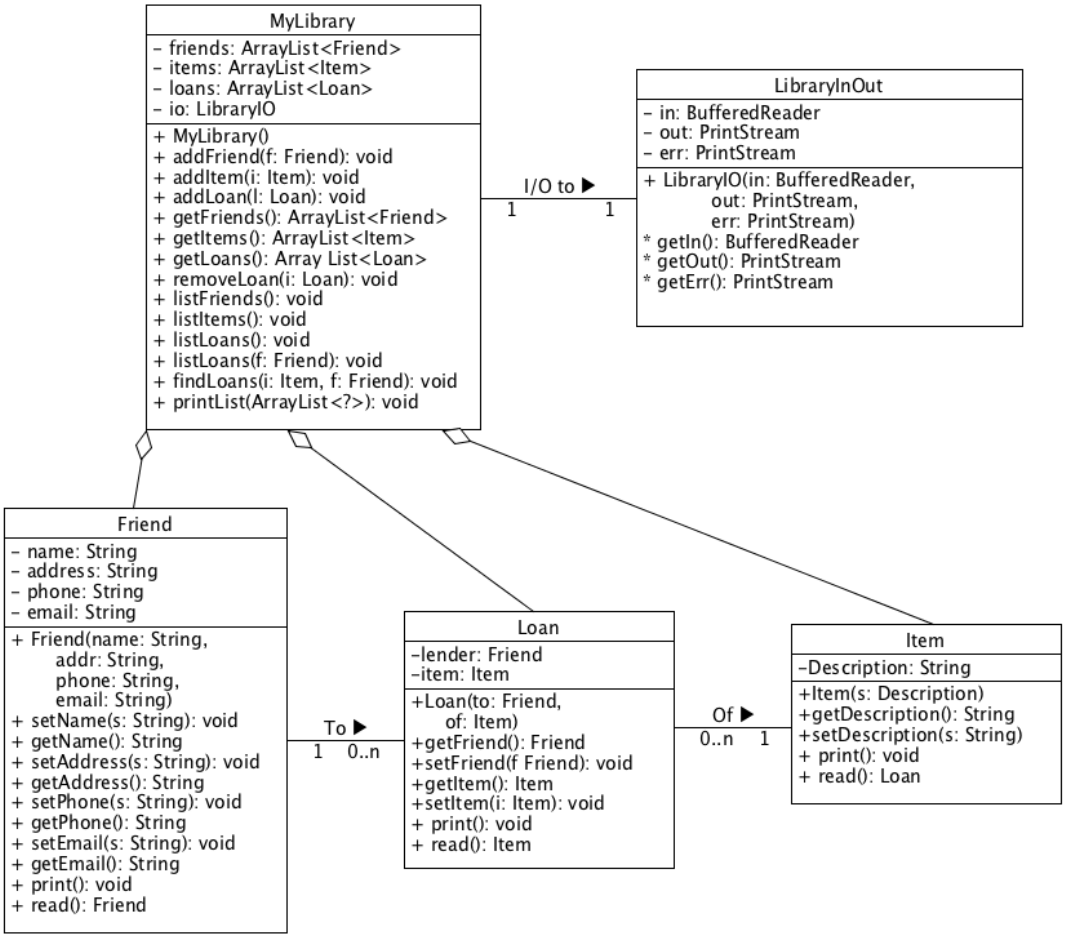

Illustration 1: Refactored Library

Notice that the function can now be named print() and read() for each of the classes. The printList() function has been moved to the MyLibrary class. It takes an ArrayList as a parameter, then prints the elements in that list. It can call the print function on each of the elements of the

list counting on the object to print itself correctly.

Printing and Reading Item and Friend

Printing an Item or a Friend is relatively easy: one simply formats the output to the stream that is passed in. For example, here is a test for printing an Item.

```
@Test
public void testPrintItem() {
    ByteArrayOutputStream stuffPrinted = new ByteArrayOutputStream();
    PrintStream out = new PrintStream(stuffPrinted);

    item = new Item("test");
    item.print(out);
    assertEquals("test", stuffPrinted.toString());
}
```

Figure 1: testPrintItem() JUnit test

As you can see, the print() function on an Item object simply prints out the description of the item. The test for reading is similar.

```
@Test
public void testReadItem() {
    BufferedReader in = new BufferedReader(new StringReader("test"));

    try {
        item.read(in, System.out);
        assertEquals("test", item.getDescription());
    } catch (IOException e) {
        e.printStackTrace();
        fail("Test threw IOException with message: " + e.getMessage());
    }
}
```

Figure 2: testReadItem JUnit test

To read in an Item object, you simply read the description. The read method will return an object created from the description that was read. Because the read method returns an object, it should be a static method: that is it is called on the class itself to create a new Item rather than being called on a particular object.

The print() and read() functions for a Friend will be a little more difficult because you need to read multiple fields to make a Friend.

Printing and Reading Loan

Printing a Loan is simple once you can print an Item and a Friend. You simply format the PrintStream to print the Friend, then the string “borrowed”, followed by the Item the Friend borrowed.

However, reading a Loan requires a bit of refactoring. When you read a Loan, you do not want to create a new Friend and a new Item when you create the Loan. Instead, you want to include a Friend from the friend's list and an Item from the items list. You will want to find the Friend in the MyLibrary friends and the Item in the MyLibrary items. However, you do not want to couple the MyLibrary class to the Loan class.

unique ID for each Friend and Item, then implement methods in MyLibrary called findFriend(String id) and findItem(string id). Then you can either store the two IDs in the Loan and pass in MyLibrary to

print. The print(MyLibrary ml) method will call findFriend() on the Loan's friendID and findItem on the Loan's itemID. Or you can store pass MyLibrary to the read() method can find the item and friend to store in the method. The second option is better because it ensures that there is a Friend and an Item to participate in the loan when the loan is created.Printing and reading a Loan is simple once you can read an print an Item and a Friend. That's why the backlog has you finish the print() and read() for the Item and Friend before you do the print() and read() for the Loan.

Hint: if you define a toString() function for a Loan, it makes testing easier. Here is my test for reading a Loan. You can also use toString() for the print() function.

```
public void testReadLoan() {
    BufferedReader in =
            new BufferedReader(
                    new StringReader(
                            "bob\nhere\n123\nbob@bob\n"
                            + "test\n"));

    loan.read(in, System.out);
    assertEquals("bob\there\t123\tbob@bob\tborrowed\ttest",
            loan.toString());
}
```

Next Week's Demo Points (30 marks)

Stories 24-30 (5 Marks Each)

- JUnit test (2 Marks)
- Implementation (3 Marks)