Final Exam

Problem 1. **[12 points (6 points each)]** In the prob1.exam package, there are classes that represent a subset of the functionality of the Library System discussed in class and implemented in the project. In the Main class, three queries are described as comments to three unimplemented methods. Pick **TWO** of the three methods to implement; the third method should have NO IMPLEMENTATION other than what has already been provided.

You may implement the methods in the Main class in one of two ways:

(1) You can create, inside the method, a lambda/stream pipeline whose return value of the expected return value for the method. Doing it this way entitles you to a maximum of 85% credit.
(2) You can add an entry in the LambdaLibrary class (provided in prob1 package), and access it in the method by calling the apply() method on your expression. This approach makes you eligible for 100% credit.

Here are the three queries.

Query A: *Given a member's checkout record, determine whether some BookCopy in the record is overdue.* NOTE: A BookCopy is overdue if
(a) it is not available, and
(b) its due date is before now

Query B: *Given a BookCopy copy and a LibraryMember mem, return true if mem has ever checked out this copy*

Query C: *Given a list of all library members, return a list, in reverse sorted order (by first name), of the full names (first name + <space> + last name) of those library members who have never checked out a book*

RULE**:** You may not modify any of the classes in prob1.exam package other than the Main class and the LambdaLibrary class.

SAMPLE SOLUTION. An example of the way solutions should be written up is in prob1.sample.

Two test classes are available in prob1.exam package: TestData and TestCode. TestData provides lists of LibraryMembers, CheckoutRecords, and Books that can be used for testing. TestCode makes use of that data to run tests on the three methods in the Main class.

DO NOT MODIFY ANY OF THE TEST CLASSES!

Problem 2. [10 points (5 points each)]. Your package prob2.exam contains three subpackages, partA, partB, partC. Each contains a designated class file (respectively PartA, PartB, PartC), along with (possibly) other classes. You will pick **TWO** of these to work on – do not attempt all three (one of the class files should remain unchanged.)

At the top of each of the designated class files, you will see a lambda expression. There are several things you will need to do with this expression.

a. Assign an appropriate type (some functional interface)
b. Express it as a method expression
c. State the type of method expression you have used
d. Express it as an inner class
e. Evaluate the lambda, the method expression and the inner class inside an evaluator() method.

There is a main method in each of the designated class files that attempts to run the evaluator method. In the body of the evaluator method, you should test your typed lambda expression, your method reference, and your inner class operation.

Each designated class provides a template for your work. You must follow this template. A sample solution is provided in the package prob2.sample. Follow the format of this sample very closely.

The lambdas provided in the three parts are:

```
PartA: () -> Math.random()
```

```
PartB: (CheckoutRecord record) -> record.getCheckoutEntries()
```

```
PartC:(Long a, Long b) -> a.compareTo(b)
```

NOTE: In partC, you may not type the lambda as a BiFunction.

Problem 3. [6 points] *Reduce.* In your package prob3.exam there are three classes: Person, FindOldestPerson, TestCode. The FindOldestPerson class contains an unimplemented static method findOldestPerson(List<Person> list). For this problem implement this method, which must return the oldest Person object occurring in the input list. Your implementation *must use the reduce method for Streams.* (If you do not use the reduce method, you will receive less than 50% credit for this problem.)

The class TestCode provides test data to test your method. You may use this class, but *do not modify it in any way.*

Problem 4. [8 points] *Exception-handling within lambda pipelines.* High school students at Fairfield High School participate in an annual word contest. Students are given 15 minutes to type into a test computer as many words as they can think of that do not begin with one of the illegal letters. In this year's contest, the illegal letters are A, B, C, E, M, N, R, S, T.

The evaluator program for student submissions runs a method adjustWords which takes the list of words created by a participant, checks that all the words are legal, turns them into lower case words, and produces a new list with these modified words. The code for this is shown in the class WordGame in your prob4.exam package. However, it has been commented out because it is implemented as a lambda pipeline in which one of the lambda implementations throws an exception which is not supported by the interface type of the lambda.

Modify the implementation of this method so that the exception is handled in one of the standard ways. Then make sure that the main method executes test data as exepcted (you will need to uncomment the code in the main method).

Problem 5. **[10 points]** A *merge function* is a function that merges two sorted lists into a single sorted list.

Example: Merging the lists [2, 4, 6], [3, 5, 6, 7] produces [2, 3, 4, 5, 6, 6, 7]

Example: Merging the lists ["Alice", "Tom"], ["Bob", "Richard"] produces ["Alice", "Bob", "Richard", "Tom"]

Example: Merging the lists [2.3, 4.5], [2,5] produces [2, 2.3, 4.5, 5]

Example: Merging the lists ["A", "XYZ", "AXTU"] and [2, 4, 6] (where the first list is sorted by word length, and in the merge operation, if a string length in the first list is the same as a number in the second list, the string comes first) produces ["A", 2, "XYZ", "AXTU", 4, 6].

Implement the most general possible merge function, assuming that input lists are in sorted order (according to a natural or a specified ordering). Your merge method should be flexible enough to correctly merge the lists shown in the examples above, and much more. Carry out your implementation in the class Merge that has been provided. The main method includes the data shown in the examples. Fill out the main method further by using this data to validate your implementation.

SCI [2 points] In the package sci, there is a text file SCI Question. Inside the text file you will see a question, asking you to relate concepts from the course to principles of SCI. Write your answer in this file and save.