MPP Standardized Programming Exam
May, 2017

This two-hour programming test measures the success of your MPP course by testing your new skill level in two core areas of the MPP curriculum: (1) Lambdas and streams, and (2) Implementation of UML in Java.You will need to demonstrate a basic level of competency in these areas in order to move past MPP.

Your test will be evaluated with marks "Pass" or "Fail." A "Pass" means that you have completed this portion of evaluation only; your professor will evaluate your work over the past month to determine your final grade in your MPP course, taking into account your work on exams and assignments. A "Fail" means you will need to repeat MPP, with your professor's approval.

There are two programming problems to solve on this test. You will use the Java classes that have been provided for you in an Eclipse workspace. You will complete the necessary coding in these classes, following the instructions provided below.

Problem 1. **[Lambdas/Streams]** In your `prob1` package, you will find a class `Problem1` that contains two static methods:

```
static List<String> elementsInJustOne(List<String> list1, List<String> list2)
static List<String> getAllFairfieldCustomers(List<Customer> list)
```

The method `elementsInJustOne` returns a list of all `Strings` that occur in exactly one of the two input lists. Below is an example of how this method should behave:

Example: If `list1 = {"A", "B", "D"}` and `list2 = {"B", "C", "D"}`, then the return list should be `{"A", "C"}`.

Hint: Consider using the static method `Stream.concat`.

The method `getAllFairfieldCustomers` returns a list of all names of `Customers` who live in Fairifield. The Customer class has been provided for you in your `prob1` package.

A `main` method has been provided that will help you test your code.

Requirements for Problem 1.

1. Your code may not contain any loops (while loops, for loops).
2. The body of each of the methods `elementsInJustOne`, `getAllFairfieldCustomers` must be a single `Stream` pipeline. You must not make use of instance variables or local variables declared in the body of either method. (Example of a local variable:
   ```
   int myMethod() {
          int x = //computation
          return x;
    }
   ```
   Here, x is a local variable. Not allowed in this problem.)
3. You may not create auxiliary methods for use in your pipeline.
4. There must not be any compilation errors or runtime errors in the solution that you submit.

<u>Problem 2</u>. [UML → Code] Translate the class diagram below into code. The diagram shows the relationships between classes in a Library System. A `LibraryMember` has a `CheckoutRecord`, which is composed of zero or more `CheckoutRecordEntries`. Each `CheckoutRecordEntry` provides information about one item (a `LendingItem`) that the `LibraryMember` checked out at some time in the past. `LendingItems` are either `Books` or `CDs`.Your solution must include every class in the diagram and all attributes and methods shown in every class. Your solution must also accurately reflect inheritance relationships, one-way associations, and dependencies shown in the diagram. Except for your implementation of the `getPhoneNums` method in `Admin`, your code must not include attributes or methods that are not directly shown in the diagram below. In your `prob2` package, you will find empty shells of all the classes that appear in the class diagram below; you must fully implement these classes, and you must not introduce any additional classes.

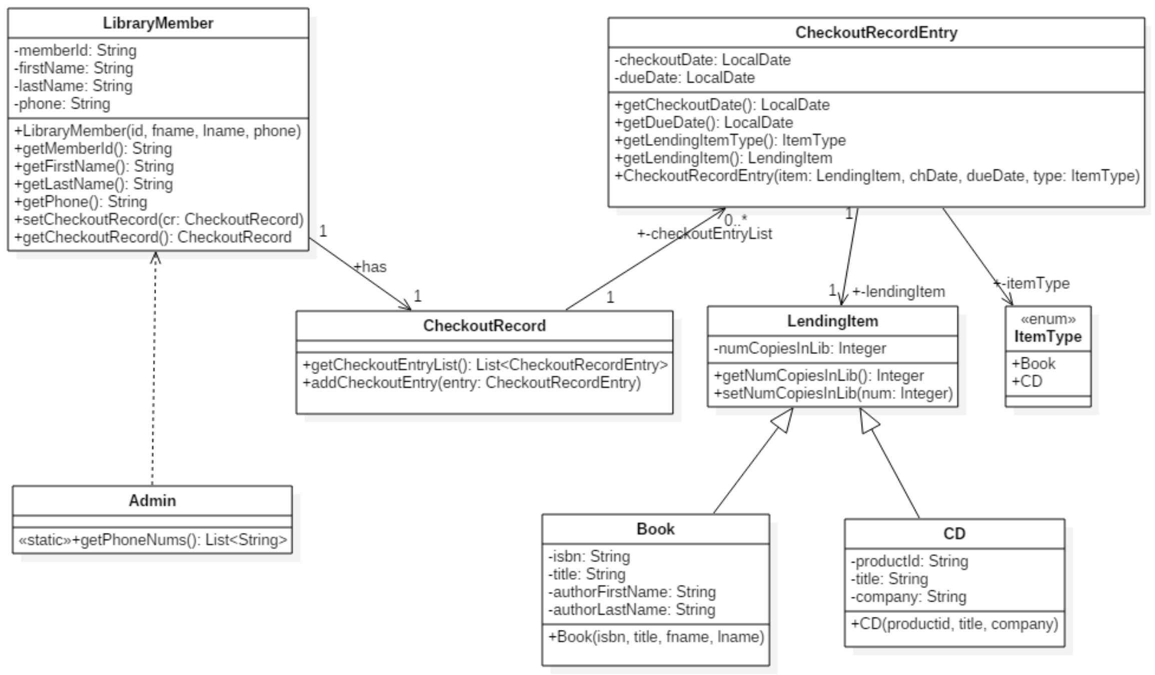
The static method

```
    static List<String> getPhoneNums(LibraryMember[] members, LendingItem item)
```

in the `Admin` class should do the following: It must return a <u>sorted</u> list of the phone numbers of those library members who have checked out, at least once, the `LendingItem item` that is passed in as input. In order to compare the input `item` with the `LendingItems` that you can find in library members' checkout records, you will have to override `equals` in an appropriate way (the `equals` method is not shown in the diagram – you will need to decide which class(es) need to override `equals`).

A class `Test` has been included in your `prob2` package (this class is not shown in the class diagram below). This class provides a `main` method with data that can be used to test your implementation. The expected output of the `main` method in `Test` is shown in the comments for that method.

Note: You do NOT need to use a stream pipeline in your implementation of `getPhoneNums` – in fact, you do not need to use any special Java 8 constructs or techniques.


LibraryMember
-memberId: String
-firstName: String
-lastName: String
-phone: String
+LibraryMember(id, fname, lname, phone)
+getMemberId(): String
+getFirstName(): String
+getLastName(): String
+getPhone(): String
+setCheckoutRecord(cr: CheckoutRecord)
+getCheckoutRecord(): CheckoutRecord
1
+has
1
CheckoutRecord
+getCheckoutEntryList(): List<CheckoutRecordEntry>
+addCheckoutEntry(entry: CheckoutRecordEntry)
1
0..*
+-checkoutEntryList
CheckoutRecordEntry
-checkoutDate: LocalDate
-dueDate: LocalDate
+getCheckoutDate(): LocalDate
+getDueDate(): LocalDate
+getLendingItemType(): ItemType
+getLendingItem(): LendingItem
+CheckoutRecordEntry(item: LendingItem, chDate, dueDate, type: ItemType)
1
1
+-lendingItem
+-itemType
LendingItem
-numCopiesInLib: Integer
+getNumCopiesInLib(): Integer
+setNumCopiesInLib(num: Integer)
«enum»
ItemType
+Book
+CD
Admin
«static»+getPhoneNums(): List<String>
Book
-isbn: String
-title: String
-authorFirstName: String
-authorLastName: String
+Book(isbn, title, fname, lname)
CD
-productId: String
-title: String
-company: String
+CD(productid, title, company)