This problem set is due Sunday, November 11 at 11:59pm, KST. Don’t go by the due date that you see on Blackboard because it is in EST. Go by the one given in this handout.

- To solve the problems below, you will be implementing classes. There is plenty of room for creativity in this course, but when I specify something, follow the specification exactly. It makes grading much easier for us.
- Each method that you write must be properly commented and attractively formatted.
- Be sure to include a comment at the top of each file submitted that gives your name and email address.

What to Submit
Submit the following files as a single zip or tar file on Blackboard. (A zip file if you are using Windows and a tar or rar file if you are using a Mac. Multiple submissions are allowed before the due date. Please do not submit .class files or any I did not ask for.

Iphone.java
UseIphone.java
Book.java
Library.java
UseLibrary.java
[any other files, if any, that are needed to run your program]
What Java Features to Use

For this assignment, you are not allowed to use more advanced features than what we have covered in Lecture 1 through Lecture 15 (Chapters 1 through 9 of our textbook plus my notes.

Partial vs. Complete Solutions

Please read what I said in PS 1 on this issue.

Naming Conventions In Java And Programming Style In General

Refer to the ones given in PS 1.

Introduction

Using account3/Account.java in lecture 15, we learned how to use static fields (e.g., nextID and sumBalance) along with their associated static methods (e.g., getNewId() and getSumBalance()). That design made sense since we want to keep track of that kind of information for a single bank. A similar situation is being dealt with in Problem 1 of this problem set. What if you want to keep track of that kind of information for multiple banks? Then, that design we used in account3/Account.java in lecture 15 would not be expressive enough. We will have to rely on an additional class such as Bank which has the capabilities of keeping track of that kind of information—one instance of Bank keeping track of information for one bank and another instance of Bank keeping track of for another bank, etc. In that case the fields and their associated methods will be dynamic, not static, right? Well, that is the kind of design that is being dealt with in Problem 2 of this problem set. So, I suggest you try Problem 1 first and then Problem 2 in that order, while trying to understand the differences between the two situations. Read the entire problem set before you start solving a problem in the set.

Pair Programming: You are allowed to do pair programming in this problem set. By pair programming I mean you work with one other person. Copying any piece of code done by your partner alone is not pair programming. In pair programming two programmers sit in front of a single keyboard/monitor and start programming, together. At any point in time during the entire programming process both are paying attention to the task at hand and whoever wants to type grabs the keyboard and types, ideally taking turns. Two people are acting as if there is only one person programming. When you submit, both partners may submit the same files. Be sure to add a comment at the top of each file that you submit stating with whom you worked. Your partner must be a student in the same class. If you choose to do it alone, that is fine too. If you violate any of the guidelines given here, you are cheating. I reserve the right to interview anyone on his or her submitted work. I allow pair programming for this problem set because it will allow you to discuss in detail between you and your partner about the design and implementation of the problems.

Problem 1 (75 points)

First, read the Introduction section before you read this problem.

Create a class named Iphone in a class named Iphone.java. As usual you should also create another class named UseIphone in a file named UseIphone.java.

In this problem you will be using static fields and methods in addition to dynamic fields and methods in Iphone.

Your end result should satisfy the following:

- Write a piece of code in the main of UseIphone to test each aspect of the requirements I describe below. As usual please do incremental development: build a little bit of Iphone and test that much in UseIphone before you add more. Repeat this process until you are done.
- An iPhone should contain at least the following attributes: name, price, display screen size (diagonal length in inches), memory capacity (in giga bytes), and whether it has a built-in compass or not. I want you to add at least two additional attributes that I did not mention. I want you to be careful in selecting the type of
each field as you design your class: int, double, boolean, String, etc. I want your decisions to be reasonable!

- Add appropriate constructor(s), getters, and setters. Don’t blindly add a getter or a setter for each field. Think about each attribute and decide whether to provide a getter and/or setter or none.

- Each iPhone that is produced, i.e., each iPhone object that gets created, must also have a serial number. To assign a unique serial number to each iPhone object, you need to come up with a way to generate a unique serial number. Include that capability in your iPhone class, and use it appropriately. You will deal with a similar situation with your library problem below when you generate a unique call number. The difference is that in the library problem you will generate them using a dynamic field in the Library class, whereas you will have to do it using a static field in iPhone in this problem.

- Your iPhone class should also be capable of remembering all the iPhone objects that have been created so far in a database and manage them. Here, the database will be implemented as an array of iPhone objects. The array will be represented as a static field here inside the iPhone class. Assume that the number of iPhone objects that will ever be created will never exceed a small number, say 10, to make your testing easier. Or, use even a smaller number like 5 as you develop your program and change it to 10 when you hand it in. Of course you will need to test it with 10 to make sure that it works with that number as well.
  
  Now add a method to the iPhone class that we can call to remove an iPhone object from that array. When you remove one iPhone object from a location in the array, you don’t want to leave a ‘hole’ in the middle of the array somewhere. I suggest you fill the hole somehow. One possibility would be to shift everything beyond the hole to the left by one position. Another solution would be to move the last one into the hole. In either case you will need to set the location where the last one was in to null. Don’t use the same technique to fill the hole in both this problem and the Library problem below.
  
  In addition to the size of the array you also need to know how many of the 10 positions in the array have been filled so far. This would be another static field that gets updated every time an iPhone object is added to or removed from the array. If you have \( n \) positions filled in so far, I assume those \( n \) iPhone objects are occupying the locations 0 through \( n-1 \), and the locations from \( n \) to the last index of the array are filled with a null value.

  Also add a method named \texttt{add} that you can call to add an iPhone object to the array. Here you will need to make a decision on how this \texttt{add} method is going to be used. When you create a new iPhone object, you may add the new object automatically into the array, in which case you would add it from the constructor. Or, don’t add it from there, but add it only if there is an explicit add request from the user, i.e., from UseIphone, by calling the \texttt{add} method. Depending on which design you choose, you will decide to make the \texttt{add} method public or private. I am inclined to suggest that you try the first option. The reason I am not insisting on one over the other is that one could argue for either one. Should the \texttt{add} method be static or dynamic?

- Add another attribute to the iPhone class for the number of phone messages that have been saved to each iPhone object. You would want to add a method to your iPhone class to be able to leave a new message to an iPhone. You don’t actually need to remember messages themselves, but just the count. So, you can call the function to leave a message without actually passing a message to leave.

- Add a method to the iPhone class that we can call to find out about the total number of messages that have been left to all the iPhone objects in the entire system. Should this be a static or dynamic method?

- Add a method named print that prints all the iPhone objects that are in the system. Should this be a static or dynamic method?

- Hints: You will need to be careful as to what needs to be static and what needs to be non-static. What needs to be private and what needs to be public. In my specification I intentionally left things open on these issues for some things so that you can think about them and make a reasonable decision yourself.

Note that UseIphone.java is given along with a sample output generated by my solution. Your solution should produce the same output. Since you will be adding more attributes of your own, your output will be slightly different. I suggest that you add another constructor with additional attribute(s) rather than changing the one I used in my solution. That way, my code in UseIphone.java would still work, which is important because it will make grading easier.
Problem 2 (75 points)

First, read the Introduction section before you read this problem. In fact, you should read Problem 1 first before you do this problem.

In Problem Set 5 you designed and implemented a Book class. This time, let’s design and implement a Library class in a file named Library.java. This class should satisfy the following:

• You may use your Book class from last problem set. You are welcome to use my sample solution if you like. You will have to add new features to the Book class as you work through this problem.

• Suppose there are multiple libraries and we will create an instance of Library to represent each library, for example, an instance for IGC Library and another for SUNYK Library. That is, you may create as many library objects as you wish in your program. Your UseLibrary class must create at least two Library objects and use them as you test your implementation.

• A library should maintain the following information:
  – It should have a name.
  – It should remember how many books are in the library at any given time. This would include all the books that have been checked out as well.
  – It should remember all the books that are in the library as well as the ones that have been checked out at any given time. You may assume that the capacity of the library is some small number such as 10 for the purpose of this problem. That will make your testing much easier. Actually you may want to use even a smaller number like 5 as you test your implementation and then increase it to 10 or even a bigger number when you feel that it has been tested adequately.

• A book in a library must have a call number assigned to it. Let’s assume that a call number assigned to a book is unique within the library. To simplify things let us assume that given a call number there is only one copy of the book with the call number in a library, checked in or out. Let’s further assume that a call number is represented as an integer. A call number is assigned to a book when the book is first added via the function add to a library.

• A book may be removed from a library via the function remove, namely the book will no longer be available in the library. When a book is removed from a library, it may create a ‘hole’ in the array data structure where you are maintaining the books. So, how do you handle the hole that gets created if you remove a book? You must fill the hole. If you have $n$ books in the array containing books, the books must occupy the locations in the array from index 0 to index $n - 1$ at all times. Also the locations in the array from index $n$ to the last index of the array must contain a null value. As I mentioned in Problem 1, don’t use the same technique to fill the hole in both this problem and the iPhone problem above.

• Given a book, a library should be able to answer if the book is in the library or not. When we ask, we should be able to ask using a call number or a title. If we ask with a call number, it will of course return a single book, but if we ask using a title, it may return multiple books. There may be more than one book with the same title. So, if we ask with a title, it should return an array of books. These two different capabilities will have to be handled using two separate functions. Let’s name these functions searchByCallNumber and searchByTitle respectively. We could use the same function/method name with overload it with different parameters. You are welcome to try it if you like.

• We should also be able to ask a library if it is empty or not. A library is empty if there is no book in the library. Let’s name this function isEmpty. When we say a library is empty, it means that there is not even a book that has been checked out.

• We should also be able to ask a library if it is full, meaning it reached its capacity and can no longer add any more books to the library. Let’s name this function isFull. A library is full if any of the arrays that are holding books (checked in or out) in the library is full.
• We should also be able to check in and check out a book. Let’s name them checkIn and checkOut respectively. The checkOut function should return the book being checked out and checkIn would not return anything useful. Let’s provide a book object when we check one in, and let’s provide a call number when we check one out.

Now that we have a class representing books and a class representing libraries, let’s build a class that tests these classes. We will call the new class UseLibrary in a file named UseLibrary.java. The UseLibrary class includes only one static method in it named main as usual. Ideally the main must test all aspects of the Library class. A good starting point would be to create an instance of Library to represent a library such as IGC Library. And create some number of books and add them to the library. Then perform various operations in some well-engineered order so that the various capabilities of the class is adequately tested. Try not to be too verbose, but add enough so that we will be convinced that your testing is adequate. In this problem, however, I decided to give a main in UseLibrary to make our grader’s task a little easier. So, use the one I give you. You don’t need to create UseBook class.

Discussion 1: As you think about the design of this problem, you will realize that almost everything in Library will be dynamic, not static. The call number generating scheme will rely on a dynamic field (e.g., nextCallID) with a dynamic method associated with the field (e.g., getNextCallId()). All the public methods such as add, remove, getNumberOfBooks, etc. will be dynamic. The main in in UseLibrary will be static, naturally. You may find some helper functions useful in Library which can be static.

Discussion 2: How many arrays would you use to maintain the books in a library? I can think of several possibilities:

• A single array containing all the books in the library. If you choose this design, you will have to add a flag (implemented as a field) to each book to remember whether the book is checked out or not at any given time.

• Two arrays: one containing all the books in the library including the ones that have been checked out; another containing only the books that have been checked out.

• Two arrays: one containing all the books that are currently in the library thus not including the ones that have been checked out; another containing the ones that have been checked out.

• Three arrays: one containing all the books in the library including the ones that have been checked out; second one containing only the checked out ones; and the third containing all the ones that are available to be checked out.

• You may even think of other possibilities.

Which would you use? It is up to you. My advice is to choose the simplest design that works.

Note that UseLibrary.java is given along with a sample output generated by my solution. By reading UseLibrary.java you can tell how these methods are used. The usage tells you whether a method is static or dynamic and its signature, which will be helpful as you design your methods. Your solution should produce the same output as mine. Here again leave UseLibrary.java unchanged so that grading will be easier. If you need to add to UseLibrary.java, add more rather than modifying existing code. In other words it would change only monotonically. The output file is given in UseLibrary.output.txt.

Advice: As usual please do incremental development: build a little bit of Library and test that much in UseLibrary before you add more. Repeat this process until you are done.

Include Book.java, Library.java, and UseLibrary.java in the zip or tar file that you submit.