Additional Problems

This 2 hour programming test measures the success of your FPP course by testing your new skill level in two core areas of the FPP curriculum: OO programming (specifically, polymorphism) and data structures. You will need to demonstrate a basic level of competency in these areas in order to move on to 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 FPP course, taking into account your work on exams and assignments. A "Fail" means you will need to repeat FPP, 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. In order to pass, you must get a score of at least 70% *on each of the two problems*.

Problem 1. **[Data Structures]** For this problem, you have been given two classes in the `prob1` package: `Employee` and `EmployeeInfo`. The `Employee` class has data fields `name` and `salary`, with getters and setters. `EmployeeInfo` is a partially implemented class, which has two parts.

- The first part of `EmployeeInfo` is the `removeDuplicates` method – this is the method you will implement. This method accepts a `List` of `Employee`s and should produce a new list of the same `Employee`s, but without duplicates.
- The second part is the `main` method which tests your `removeDuplicates` method. The `main` method passes in a `List` of `Employees` from the `TestData` class (called `originalList`) and then compares the output of your method with the list `dupsRemoved` in `TestData`. The `dupsRemoved` list is the list of `Employee`s contained in `originalList` with all duplicates removed, so it is the correct output for the `removeDuplicates` method. The `main` method then tests whether the return list of your `removeDuplicates` method is the same as the `dupsRemoved` list; it checks this by calling the utility method `listsAreEqual`, located in `prob1.util.Util`.

For this problem, you must implement the `removeDuplicates` method. You may asssume that the input list for `removeDuplicates` is not `null` (so you do not need to test for `null` in your code).

In your implementation of `removeDuplicates`, you *are not allowed to* use any implementation of Java's `Set` interface.

Your `removeDuplicates` method will, at each step, need to determine whether two given `Employee` objects are equal. To perform this comparison, you *must* override Java's `equals` in the `Employee` class. *Note*: Two `Employee` objects are considered to be equal if and only if they have the same `name` and `salary`.

To get full credit for this problem, there are two requirements:

1. The output of your `removeDuplicates` method must be a list that is identical to the `dupsRemoved` list in `TestData`

2. The utility method `Util.listsAreEqual` must return `true` when `TestData.dupsRemoved` and the list obtained from calling your `removeDuplicates` method are both passed in as arguments to `Util.listsAreEqual`.

In developing a solution, you are not allowed to modify the `TestData` or `Util` classes in any way, and you are not allowed to modify the `main` method in `EmployeeInfo`. You are allowed to *add* code to `Employee`, but *not modify* existing code in that class.

Important! To clarify any possible confusion: The only code you need to write is to implement the method `removeDuplicates` in the `EmployeeInfo` class and to override `equals` in the `Employee` class. You do *not* need to write any additional code to test whether lists are equal (this has already been done for you).

Problem 2. [Polymorphism] In the `prob2` package of your workspace, you are given four fully implemented classes and a `Main` class: `Bank, CertificateDeposit, CheckingAccount, SavingsAccount`. The classes `CertificateDeposit, CheckingAccount, SavingsAccount` represent different types of accounts that a bank customer may have. Each contains a method `computeTotal` which provides a value for the current account balance; in its computation, `computeTotal` factors in the interest (if there is any interest) for the account.

The `Bank` class is used for accumulating information about multiple accounts – in particular, the `Bank` class can be used to store several accounts and then compute the total balance across all of these stored accounts, using the method `computeBalanceSum`.

The `Bank` method `computeBalanceSum` has not yet been implemented. You must implement this method so that summation of account balances is done *polymorphically*. This requirement implies that your implementation *does not check* the runtime types of the accounts in the `accountList` of `Bank`. To satisfy this requirement, you *will need to create and use an interface*, which should be added to the `prob3` package and implemented by some of the classes in this package.

You are allowed to make changes to the generic type of the `accountList` in `Bank`, and also to the type of the argument in the `Bank` method `addAccount`.You are not allowed to modify the code in the `Main` class in any way. You are allowed to make changes to the class declaration of the classes `CertificateDeposit, CheckingAccount, SavingsAccount`, but you are not allowed to modify the `computeTotal` method in any of these classes. You are not allowed to remove the `final` keyword in the declaration of the `Bank` class.

Test your work by running the `main` method of `Main`. Note that this `main` method relies on an extra class `RandomNumbers` in `prob2.util`; the `RandomNumbers` class should not be modified in any way