Lesson – 11 Hash Table & Polymorphism Concepts

Note : The given two problems are asked in the previous batch Common Programming Test.

Problem 1. [Data Structures – Hash Table & ArrayList] In your prob1 package, you will find two classes, Employee and EmployeeAdmin. A Main class is also provided that will make it convenient to test your code.

The Employee class has been fully implemented. It has three fields: name, salary, and ssn (which stores a social security number). Employee provides getters and setters for each of these fields.

The EmployeeAdmin class is intended to provide reports about Employees. For this problem, the EmployeeAdmin class has just one static method, prepareReport, which accepts a HashMap table and a List socSecNums as arguments. The HashMap matches employee social security numbers with Employee objects. The List contains some employee social security numbers, represented as Strings.

Your method prepareReport must produce a list of all Employees in the input table whose social security number is in the input list socSecNums and whose salary is greater than $80,000. In addition, this list of Employees must be sorted by social security number, in ascending order (from numerically smallest to numerically largest).

The main method in the Main class provides test data that you can use to test your code.

Here is an example of how the method prepareReport should behave: In the input table, you see four entries: The first entry associates with the ssn "223456789" the Employee object ["Jim", 90000, "223456789"]. There are three additional entries in table. The list socSecNums, also provided, contains four social security numbers.

```
table:
   "223456789"   ["Jim",   90000,   "223456789"]
   "100456789"   ["Tom",   88000,   "100456789"]
   "630426389"   ["Don",   60000,   "630426389"]
   "777726389" ["Obi", 60000, "777726389"]

socSecNums:
    "630426389",  "223456789"  , "929333111",  "100456789"
```

When we scan the list socSecNums, and use these values to read the table, we find only three of the employees: Jim, Tom, Don. We also notice that only Jim and Tom have salaries greater than $80000, so only these two Employees will be returned in our final list. We then sort this list of two `Employees` (Jim and Tom) according to the order of their social security numbers. The final output should be :

```
   ["Tom", 88000, "100456789"],  ["Jim", 90000, "223456789"]
```

Requirements for this problem.

(1) Your list of `Employees` must be sorted using a sorting method in Java's `Collections` class.

(2) Ordering of `Employee`s must be determined by a `Comparator`, which you must define yourself (and include in your workspace). Your `Comparator` should follow this rule: Given `Employee`s `e1` and `e2`, `e1` should be considered "less than" `e2` if the social security number of `e1` precedes the social security number of `e2` (in the natural ordering of `String`s). *Note*: You may assume that no two employees provided in the input table have the same social security number.

(3) Your return list of `Employee`s must not contain `Employee` objects whose social security number is not on the input list `socSecNums`. Note also that there may be social security numbers in the input list `socSecNums` that do not belong to any of the `Employee` objects in the table.

(4) Your list of `Employees` must not contain any `null`s.

(5) You may not modify the `Employee` class in any way.

(6) There must not be any compilation errors or runtime errors in the solution that you submit.

<u>Problem 2</u>. [Polymorphism & ArrayList] In the `prob2` package of your workspace, there are two subpackages: `prob2.incorrect` and `prob2.solution`. Both packages contain an `Employee` class, as well as classes for three different types of bank accounts (`RetirementAccount, SavingsAccount, CheckingAccount`). An `Employee` has instance variables `id` and `accounts` (which is a list of accounts). Each employee account will be one of the three account types mentioned above. There is also an `AccountManager` class containing a static method `computeAccountBalanceSum`, which takes as input a list of `Employee` objects; for each such `Employee` object, it extracts the balance from each of the accounts in the list of accounts contained in that `Employee`, and adds them to a running `sum` variable; finally, `computeAccountBalanceSum` returns the final value of `sum`.

In the package `prob2.incorrect`, all the code is correct and the total sum of balances that is computed by the `AccountManager` is correct; however, the implementation in this package is of very low quality because polymorphism has not been used.

The objective of this problem is to rewrite this code so that polymorphism is used. All the classes in `prob2.incorrect` have been copied into the package `prob2.solution`. You must make the necessary modifications to the classes you find in `prob2.solution` so that computation of total sum of balances is still correct, but computation is done polymorphically. An abstract class `Account` (unimplemented) has been provided for you to assist in your polymorphic implementation.

In each of the packages `prob2.incorrect` and `prob2.solution`, a `Main` class is provided that can be used to test the `AccountManager.computeAccountBalanceSum` method.

Requirements for this problem.

1. All Lists in your solution package must have an appropriate type (for instance,

   `List<Employee>` rather than just `List`).
2. Your implementation of `computeAccountBalanceSum` in `AccountManager` must

   correctly output the sum of the balances of all accounts in all the `Employee` objects passed in as an argument.
3. Your implementation of `computeAccountBalanceSum` must make correct use of polymorphism .
4. *None of the code in the prob2.incorrect package may be modified!*
5. You are allowed to modify declarations of the different bank account classes, but the *final* keyword used in these classes may not be removed.
6. There must not be any compilation errors or runtime errors in the solution that you submit.