Lesson – 11 Hash Table & Polymorphism Concepts

Problem 2. [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 `Employee`s. 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 `String`s.

Your method `prepareReport` must produce a list of all `Employee`s 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 `Employee`s 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 `Employee`s (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 `Employee`s 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.