

# COMP 250 Assignment 4

**Posted:** Wednesday, March. 30, 2011

**Due:** Sunday, April 10, 2011 23:59

**Late submissions will be accepted up to one week late (penalty of 10 points)**

## General Instructions

The T.A.s handing this assignment are Faiyaz and Juan (Question 1) and Jonathan (Question 2). Their office hours and location will be posted on the public web page.

Do not change any of the given code. Only add code where instructed. Do not change the package name, as this slows down the grading.

The starter code that is provided includes comments before each method that allow you to generate Javadoc. You are encouraged to learn Javadoc but it is not required.

You are expected to comment your code so that the grader can easily follow what the code is doing. Points will be removed for poor style (improper indentation, non-existing or unhelpful comments.)

## Question 1 Hash Tables (40 points)

In this question, you will get “hands on” experience with hash tables by implementing the basic methods. You are given a **MyHashTable** class with code stubs. You are also given a test class which uses a hash table to store a list of songs. You are required to implement the following:

### a) (5 points)

The methods in the nested class **HashEntry**. The HashEntry class represents a single `<key,value>` pair in the hash table.

### b) (10 points)

The **HashIterator()** nested class which implements the `java.util.Iterator <E>` interface. Implement the constructor, and the **hasNext()** and **next()** methods. You may add any private variable to this class if needed. Your **next()** implementation must avoid throwing any exception if there is no more element. Note that, you do *not* need to implement the **remove()** method.

Hint: The iterator code for BST shown in lecture 21 should give you an idea for creating a simple iterator for a hash table.

c) (20 points)

The constructor, **clear()**, **containsKey()**, **containsValue()**, **get()**, **isEmpty()**, **iterator()**, **keys()**, **put()**, **remove()**, **size()**, **values()** methods. For the detailed behavior of these methods, please consult the javadoc and given comments in the code.

d) (5 points)

The **rehash()** method. This method is called by the **put()** method whenever the load factor exceeds the **MAX\_LOAD\_FACTOR** threshold. **rehash()** doubles the number of available buckets and reorganizes the current elements in the hash table.

Your hash table implementation should use the separate chaining approach discussed in the lecture on hashing. You may use your iterator for the implementation of other methods.

Hand in the file **MyHashTable.java** .

## **Question 2 Object Oriented Design (40 points)**

This question will give you some practice with inheritance and polymorphism in Java.

a) (20 points)

You are provided with a class **Employee**. Write two classes **Salesperson** and **Manager** which extend **Employee**. A **Salesperson** is an **Employee** whose pay is the sum of a base salary and a commission, where the commission is a percentage of the salesperson's **sales**. A **Manager** is an **Employee** whose pay is the sum of a base salary and a percentage of the total sales of all the **Salesperson** employees of that company.

For both classes **Salesperson** and **Manager**, you need to override the **getPay()** and **toString()** methods.

The **Salesperson** class must have a field

- **private double sales;**

with a getter and setter method. The commission rates for two types of employee are determined below.

b) (20 points)

You are provided with an abstract class **Employer**. Write a class **RetailStore** which extends **Employer**. The employees of a **RetailStore** include managers and salespersons (see above) as well as other general employees. For any retail store, all managers have the same base salary and commission rate, and all salespersons have

the same base salary and commission rate. The employees of a retail store also have employee numbers (an **id** field), which correspond to the order in which employees are added.

The **RetailStore** class should have the following fields:

- **private int** managerBaseSalary;
- **private double** managerCommissionRate;
- **private int** salespersonBaseSalary;
- **private double** salespersonCommissionRate;

and the following methods:

- getters and setters for the fields
- **addEmployee( Employee name)** which overrides this method from **Employer**.
- **setSales(Salesperson name, double sales)** which specifies the sales for this salesperson
- **getTotalSales()** which returns the total sales of all Salespersons of the store
- **getTotalPay()** which returns the total pay of all Employees of the store (including commission for managers and salespersons)
- **toString()** which overrides this method from the **Employer** class

## Other Specifications:

- You are given a class **TestRetailStore** which has a main method. This test class should produce the output below. The values in each line of output are important, but the exact formatting is not. (Your classes also will be tested using a different test class):

```
SweetDeals : 6 employees
name: Pascal, ID: 1, Employee type: Manager
pay: 5440.0 ( base = 5000, commission = 440.0 )
name: Oscar, ID: 2, Employee type: Salesperson
pay: 1500.0 ( base = 1000, commission = 500.0 )
name: Ming, ID: 3, Employee type: Salesperson
pay: 2000.0 ( base = 1000, commission = 1000.0 )
name: Patricia, ID: 4, Employee type: Salesperson
pay: 1700.0 ( base = 1000, commission = 700.0 )
name: Danielle, ID: 5, Employee type: general
pay: 3000.0
name: Raheem, ID: 6, Employee type: general
pay: 2000.0
Total sales is 22000.0
Total pay is 15640.0
```

- Submit files **Salesperson.java**, **Manager.java**, **RetailStore.java**.