Assignment for Software Development 1, WS 2012/13 — Assignment 8

Name: _____ Teaching Assistant: _____

Student ID (Matr.Nr.): _____ Points (max. 24): _____

Group: _____ Deadline: Wed, Jan. 16th, 2013 12:00 noon

Instructor: _____ Editing time: _____

Question 1: Document Word Frequency — 12 Points

Implement a program that will read in a document of text (words) in order to perform some basic operations on the document. The first operation is to find the **set of unique words** in the document. The second operation is to **sort in ascending and descending order** the words based on their frequency of occurrence in the document.

You should organize your program in the following 3 classes.

1. The first class is called **`Element`** and contains two features: a word (type `String`), and a frequency counter of the word (type `int`).
2. The second class is called **`Document`** and contains a vector (see `Vector`-class) of elements (type `Element`). The **`Document`** class should accept an input file (filename of type `String`) in the constructor and handle the processing of the corresponding text file (file should be placed in your working directory; see also hints below). The document class shall support, amongst others, the following operations:
   - `getWords`: Return the list of unique words from the document,
   - `getWordList(order)`: Sort the list of unique words based on their frequency of occurrence; for 'order' implement sorting in ascending and descending order and define an appropriate parameter to select between the order types.
3. The third class is the **`DocumentTest`** class, in which your main method and additional methods to process data should be placed in order to test your implementation (i.e., **`Document`** class). The **`DocumentTest`** class will, for example, support the following operations:
   - `printWords(...)`: Print the list of unique words to the console,
   - `printElements(...)`: Print the list of elements (i.e., unique words as well as their frequencies).

   To read in the text file you can use the `Scanner`-class. For details see the Java documentation: http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

Examples:

- You should at least use the accompanying text file '`alltitles.txt`' as an example to test your implementation.
- In the following, testing of the `Document`-class on this file is shown. In detail, the 5 most frequently occurring words from '`alltitles.txt`' are printed to the console in descending order. For this, the `printElements`-method is called (and within the method, `getWordList(...)` might be used) to generate the following result:

```
of  46
the  37
in  15
and  14
by  8
```

- This result indicates that the most frequently occurring word in the document 'alltitles.txt' is the word "of", which occurs 46 times. The next most frequently occurring word is "the", occurring 37 times, and so on.

Hints:

- In order to read a text file, you can use the `Scanner`- and `FileReader`-classes (for details on reading files see the lecture material).
- A method in which the text file is opened should handle an exception in case the file was not found. For example, if you are using a FileReader to open the file, then exception handling could be implemented as folllows:

```
Filereader r = null;
try {
   // Open file and read contents
   r = new FileReader(fileName);
   // Process file...
} catch (FileNotFoundException e) {
   System.err.println("File not found.");
}
```

- You also need to close the text file when you have finished reading it. The `Scanner`-class has a method called `next()` which can be used to read words. You may also need to use the `hasNext()` and `useDelimiter()` methods.

Write the program in Java. Provide all the material requested below at the very bottom ("Requested material…")

Question 2: Company Employee List **12 Points + 6 Extra Points (see Question 2a)**

Develop a program named **`Company`** that manages a list of employees (class `Employee`) as a **double linked list** and test its functionality by implementing a separate **`CompanyTest`** program. Your set of classes (mainly `Company`) will have to support the following operations:

- Adding a new employee to the company,
- Looking for an existing employee based on their last name only,
- Returning the list of employees with a salary greater than X (sorted by ascending employee number),
- Returning the list of employees with a salary greater than X (sorted by descending employee number),
- Printing to the console the entire list of employees in ascending order of employee number (i.e., call the `toString()`-method of `Company`),
- Printing to the console the entire list of employees in descending order of employee number (toString() cannot be used here; this means that you have to iterate over all employees starting from `tail`, and call the `toString()`-method on each `Employee` object),
- Deleting an employee from the company.

1. Start by implementing the `Employee`-class. An employee should have at least the following attributes: a name (firstname, lastname), an employee number, and a salary. The employee number should be a random 10-digit number generated in the `Employee` constructor. For simplicity, an employee number is not necessarily unique (i.e., you do not need to check for two identically generated employee numbers).

- Supply appropriate **constructors and public getter/setter methods** for the **`Employee`**-class. Provide, for instance, methods like `getFirstName`, `setLastName`, `getEmployeeNum`, `getSalary`, `setSalary`, etc. with the appropriate parameters.
- Override the public method `toString()` to return a textual representation of an employee as shown in the following example:

```
@Override
public String toString() {
    // should return the following string for an employee:
    // #1234567890, FirstName LastName [salary=0.0]
}
```

2. Within the **`Company`**-class, a **list of employees** should be organized in a **double linked list, whereby the list is ordered based on the employee number** (this means, that you have to add new employees at the correct position in the list!)**.**
   - Please remember to define and manage **`head`- and `tail`**-references in the **`Company`**-class.
   - **Do not use existing Java list implementations (e.g., the `LinkedList`-class) for this question as the goal of this problem is to test your fundamental knowledge of linked lists!**
   - Also override the `toString()`-method in the class `Company` which should return a string representation of all the employees sorted in the default order (ascending employee number). You may implement the method as illustrated in the pseudo code snippet:

```
@Override
public String toString() {
   StringBuilder builder = new StringBuilder();
   // for all employees {
   //     builder.append(employee.toString());
   // }
   return builder.toString();
}
```

Hints:

- Please adhere to the principles of object oriented programming, i.e. separation of data structures/data models and (user specific) processing, i.e., print-methods must not be implemented in the data (`Employee`, `Company`) classes!
- You should create the following three classes:
  - An `Employee` class. The employee object stores the name, employee number, salary, etc.

```
// The Employee class
public class Employee {
    private String firstName;
    private String lastName;
    // set once in the constructor and leave then unchanged
    private long employeeNum;
    private double salary;
    ... // additional attributes
    Employee next;
    Employee prev;

    // Employee constructors
    public Employee(...) {
        ...
    }
```

```
    // getter and setter methods
    public type getXXX () {
        ...
    }
    public void setYYY (param) {
        ...
    }
}
```

- A `Company` class that manages a (double linked) list of all employees.
- A `CompanyTest` class with a main method to creates a company object and manipulates the company (add employee objects, delete employee objects, print the list of employees in reverse order, etc.).

Write the program in Java. Provide all the material requested below at the very bottom ("Requested material…")

Question 2a: Company Employee List - Encapsulation **6 Extra Points**

As an **experienced programmer** you might have indicated that the proposed class/object structure does not really follow (for simplicity reasons) object oriented principles. If you are experienced enough, or have fun in experimenting with Java, you can implement Question 2 based on the following underlying class structure:

```
class Company {
   EmployeeList l;
}

class EmployeeList {
   EmployeeNode head, tail;
}

class EmployeeNode {
   EmployeeNode next, prev;
}

class Employee {
   private String firstName;
   private String lastName;
   ...
}

class TestCompany {
   public static void main(...) {
      ...
   }
}
```

Please note that Question 2a is optional, and that you have **either** hand **in Question 2 or Question 2a**.

Requested material for all programming problems:

- **For each exercise, hand in the following:**
  - **a) The idea for your solution written in text form.**
  - **b) Source code (Java classes including English(!) comments).**
  - **c) Test plan for analyzing boundary values (e.g., min. temperature allowed, maximum number of input, etc.) and exceptional cases (e.g., textual input when a number is**

required). State the expected behavior of the program for each input and make sure there is no "undefined" behavior leading to runtime exceptions. List all your test cases in a table (test case #, description, user input, expected (return) values).

d) The output of your java program for all test cases in your test plan.

- Pay attention to using adequate and reasonable data types for your implementation, check the user input carefully and print out meaningful error messages.

Advice for user input: While no longer mandatory it is still recommended to use the class Input.java for user input (read operations) in your program.