Software Development I – Exercises **Winter Term 2012/2013**

Übungen zu Softwareentwicklung 1 **Assignment 7**

Name: _____ Teaching Assistant: _____

Student ID (Matr.Nr.): _____ Points (max. 0): _____

Group: _____ Deadline:Wed, January 9th, 2013 12:00 noon

Instructor: _____ Editing time: _____

Question 1: MyOwnString **12 points**

Implement a class `MyOwnString`, which manages a string as array of characters (e.g., `char[] data`).

The class should at least have the following constructors and methods:

```
// Constructors -------------------------------------------------------------

/* creates an empty string */
public MyOwnString() {...}

/* copies the content of rawData and sets the length depending on the number
   of characters in the existing array */
public MyOwnString(char[] rawData) {...}

/* initializes the instance with characters of string s */
public MyOwnString(String s) {...}

/* creates a new string initialized with the content of MyOwnString s */
public MyOwnString(MyOwnString s) {...}


// Methods ------------------------------------------------------------------

/* returns the length of the string */
public int length() {...}

/* returns the saved string as an instance of java.lang.String */
public String toString() {...}

/* returns the string starting at the character with index start (inclusive)
   to the character with index end (exclusive); returns null if any of the
   indexes given is not valid for the string */
public MyOwnString substring(int start, int end) {...}

/* adds the string s to the end of the existing string */
public void concat(MyOwnString s) {...}

/* adds the string s2 to the end of s1 and returns the result as MyOwnString */
public static MyOwnString concat (MyOwnString s1, MyOwnString s2) {
}

/*compares the string to s lexicographically and returns a negative integer if
  this string object lexicographically precedes the argument s, a positive
  integer if this string object lexicographically follows the argument s, and
  zero if the two given strings are equal
  Note: thereby only letter character A-Z and a-z shall be considered for the
  comparison; any other character shall be ignored */
public int compareTo(MyOwnString s) {...}
```

```
/* returns the index within this string of the first occurrence of the specified
   string argument s, returns a negative value if the substring was not found */
public int indexOf(MyOwnString s) {...}

/* eliminate all leading and tailing white-spaces of the string; returns the
   number of white-spaces eliminated */
public int trim() {...}
```

Examples

```
MyOwnString a = new MyOwnString(new char[]{'H','e','l','l','o'});
MyOwnString b = new MyOwnString("World!");
a.concat(b);
System.out.println(a); //returns "Hello World!"
```

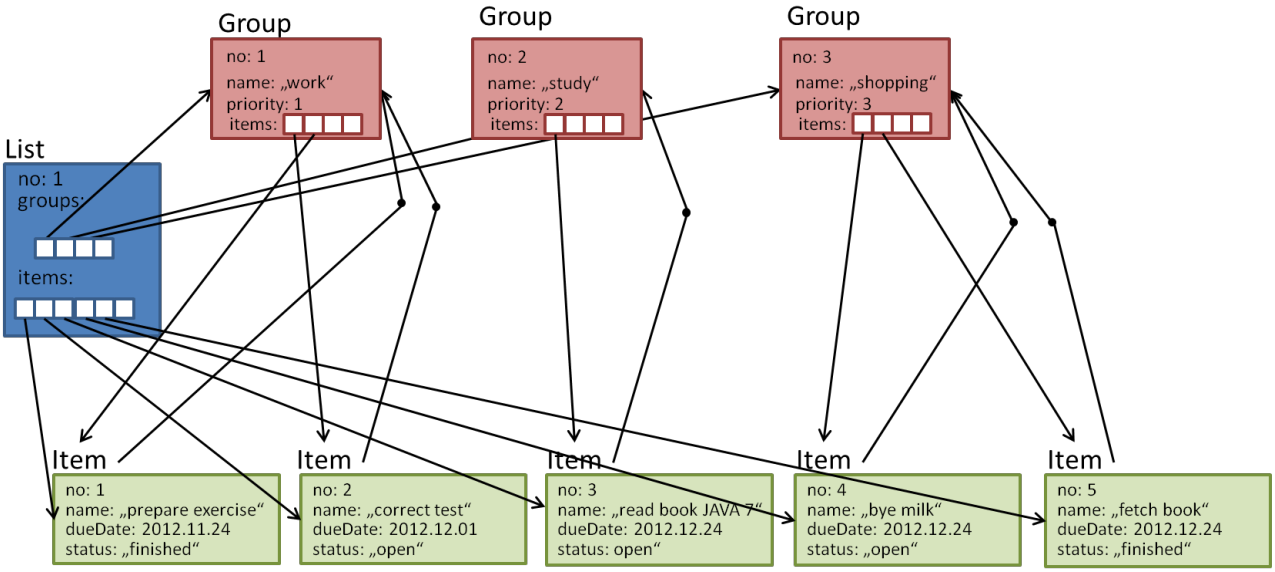
Note

- The length of the internal character array shall vary depending on the need.
- Maintain the length of the string in a **separate dedicated instance variable**.
- For the sake of this exercise you shall implement all methods on your own thereby not relying on existing methods of Java.
- Care for error checking. Consider what will happen if e.g. you call `s.substring(0,20)` for a string with only 10 characters.
- **Test each of your methods separately, providing a test plan and a test output. For that, realize a `main` method comprising all relevant tests during a test run.**

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

Question 2: ToDo-Entries **12 points**

Implement a "ToDo"-list (class `List`) which shall comprise a series of "ToDo"-items (objects of the class `Item`; stored in the array *items*) and a series of groups of "ToDo"-items (objects of class `Group`; stored in the array *groups*).



Each `item` (class `Item`) consists of a *textual description* (`String`), *a due date* (`java.util.GregorianCalendar`), a *status* (`"open"`, `"started"`, `"finished"`; data type `String`) and belongs to one `group` (class `Group`) (e.g., `"work"`, `"study"`, `"shopping"`, …).

Each `group` (class `Group`) has a *name* (uniquely identifying the group, data type `String`) and a specific *priority* (value '1' denoting the highest priority, data type `int`) and, most importantly, a series of items (objects of class `Item`; stored in the array `items`) belonging to this group.

Note

- All items, lists and groups shall receive a unique ascending number (*int*) for unique identification, which is stored and printed for each object.

Class `Item`

The class `Item` shall allow for items consisting of a textual *description*, *a due date* and a *status* (`"open"`, `"started"`, `"finished"`). An item must belong to exactly one group.

Print functionality

The class `Item` shall allow for printing an item.

```
/* Prints the item and the information about the group it belongs to. */
print();
```

Example output

```
#7, prepare exercise, work, 2012.11.24, open
```

Hints

- Use `java.util.GregorianCalendar` to realize the due date of an item.
- Date creation:
  ```
  Calendar date1 = new GregorianCalendar(2013, 0, 1); // January 1st, 2013
  ```
- Date comparison:
  ```
  if (date1.compareTo(date2) < 0) ... // date1 is before date2 ...
  ```
- Print a date:
  ```
  System.out.println(aDate.get(Calendar.YEAR)+"."        // year
                     +(aDate.get(Calendar.MONTH)+1)+"."  //  month
                      +aDate.get(Calendar.DAY_OF_MONTH)); // day
  ```
- See the following web page for more information on 'Date':
  ```
  http://docs.oracle.com/javase/7/docs/api/java/util/GregorianCalendar.html
  ```

Class `List`

The class `List` shall allow referring to its items and its groups and allow for inserting new items and groups.

Inserting

The class `List` shall allow for inserting of items and groups and thereby maintain its groups and items in special order.

```
/* Inserts an item into the list and into the according existing group within
   the list if and only if the item is not yet contained within the list and the
   specified group exists. The items of a list shall be maintained to be ordered
   by due date, then status and then name. Inserting an item into a list
   requires inserting this item also in one of the maintained groups. In case no
   insertion can be made (also due to space limitations of the array) false
   should be returned, otherwise true. */
boolean insertItem(Item item, Group group) {...}

*/ Inserts a group into the array named groups of a list if not already present
   in the list. Groups shall be maintained sorted according to their priority
   (starting with 1) and their name.
   In case no insertion can be made (also due to space limitations of the array)
   false should be returned, otherwise true. */
boolean insertGroup(Group group) {...}
```

Print functionality
The class `List` shall allow for printing its items, either group wise or as ungrouped list.

```
/* Prints the items of the list according to the order of items. */
void printAll() {...}
```

Example output
```
#1, prepare exercise, work, 2012.11.24, finished
#2, correct test, work, 2012.12.01, open
#3, read book JAVA 7, study, 2012.11.24, open
...
```

```
/* Prints the items of the list by group according to the order of groups and
   items therein. */
void printInGroups() {...}
```

Example output
```
#1, work, 1
--------
#2, correct test, work, 2012.12.01, open
#1, prepare exercise, work, 2012.11.24, finished

#2, study, 2
--------
#3, read book JAVA 7, study, 2012.11.24, open
...
```

Hints
- Use appropriate arrays of fixed length to maintain the necessary lists of groups and items; avoid errors during runtime. Dynamic growth of the arrays does not need to be foreseen.
- Take care to insert an item, both, in the list of items and in the according group.
- Maintaining the order when inserting an item means that the appropriate position in the array is found according to the prescribed order shifting items ordered behind the item inserted back in the array.

Class `Group`
The class `Group` should hold a list of its items (*items*) and provide methods, which allow inserting, sorting, and printing of its items.

Inserting
The class `Group` shall allow for printing a group and its items whenever an item is inserted into its respective list.

```
/* Inserts an item into the group storing the item in the group's items-array.
   Assure that an item is only ones contained within a group. The items within a
   group shall be resorted according to the last applied order. In case no
   insertion can be made (also due to space limitations of the array) false
   should be returned, otherwise true. Note that it is intended that this method
   is called by List.insertItem() only since otherwise inconsistencies may
   occur, however, no precautions need to be taken for that. */
boolean insert(Item item)
```

Sorting
The class `Group` shall allow for explicit (re-)sorting of its items according various orders, e.g., by status (`sortByStatus()`) or by due-date (`sortByDueDate()`) .

```
/* Resorts a group's items to order by 1) status, 2) due date and 3)
   description. */
void sortByStatus() {...}
```

```
/* Resorts a group's items to order by 1) due-date, 2) status, and 3)
   description. */
void sortByDueDate() {...}
```

Printing

The class `Group` shall allow for printing of the group and its items in the current order.

```
/* Prints the items of the group according to the order of items. */
void printAll() {...}
```

Example output

```
#1, work, 1
--------
#2, correct test, work, 2012.12.01, open
#1, prepare exercise, work, 2012.11.24, finished
```

Hints

- Use appropriate arrays of fixed length to maintain the necessary lists of items; avoid errors during runtime. Dynamic growth of the arrays does not need to be foreseen.
- Take care to remember the way the items of the group were sorted last to be able to resort the list when a new item is inserted.

Test program

Demonstrate the functionality of your program through an appropriate initialization of lists, groups and items, and execution of the required methods. For this use at least two lists, three groups and seven items.

Notes (for all classes)

- Find and realize appropriate constructors for the different classes.
- Find and realize appropriate additional (private) methods to provide the required functionality.
- Provide appropriate return values to deal with the requirements (e.g., existing group when inserting an item in a list).

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

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., minimal temperature allowed, maximum number of input, etc.) and exceptional cases (e.g., textual input when a number is required, etc.). 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 use adequate and reasonable data types and meaningful English variable names for your implementation, check the user input carefully and print out meaningful error messages.