Assignment 3: Task Management System

Note
• This is an individual assignment; all the work must be your own.
• Download the skeleton from Canvas and finish the implementations using the skeleton.
• Upon completion, compress the directory of the project and upload a zip file to Canvas.
• The due time is 11:59 pm on May 8. Late submission will be penalized by 20% per day.
• This assignment accounts for 15% of the total course grade.
• Email to cwangch@cse.ust.hk if you have any questions.
• Cheating/plagiarism will be caught and punished HEAVILY!
University's Honor code: http://ugadmin.ust.hk/integrity/honor.html
Penalties for Cheating: http://ugadmin.ust.hk/integrity/student-5.html

1. Background

In this assignment, you are required to design a task management system. You need to be familiar with the usage of lambdas and higher functions to implement the expected functionalities.

2. Task Management System

The task management system is expected to manage three kinds of tasks, namely READING, CODING, and WRITING. Each task can contain one or more tags, which might be used for the further management.

Tasks in the management system

String id, title, description;
TaskType type;
LocalDate createdOn;
Set<String> tags;

We have provided the implementation of the class *Task*. You can use it to construct a task object with various constructors and get the attribute values by its member functions.

The task management system maintains a list of the tasks. Basically, the system should support the following functionalities:

1. Add a task object to the system.
2. Find and return all unique tags from all the tasks and sort them by the tag content ascendingly.
3. Find the tasks satisfying a specific condition and sort them by the IDs ascendingly:
    a) The task is in a specific type;
    b) The task has a specific tag;
    c) The title of the task contains a specific keyword as a substring;
    d) The description of the task contains a specific keyword as a substring;
    e) The creation time is before a given time.
4. Count the numbers of the tasks satisfying a specific condition in 4(a)~4(e).

5. Return the Top-N tasks satisfying a specific condition in 4(a)~4(e), where the tasks are sorted according to the ascending order of the ID.

6. Remove the tasks satisfying a specific condition in 4(a)~4(e).

Particularly, the order of the ID is determined by the order of the string content inherently.

3. Implementation Notes

To support the above functionalities, you are required to implement the following methods in the java class *TaskManagementSystem*.

Method	Method Description
void appendNewTask(Task t)	Add a task object
List<String> findAndSortAllUniqueTags()	Find and sort all unique tags
List<String> findTasks(Predicate<Task> p);	Find the tasks satisfying the condition p. The result is sorted by IDs.
int countTasks(Predicate<Task> p);	Count the tasks satisfying the condition p
List<String> getTopNTasks(Predicate<Task> p, int N);	Find the top-N tasks satisfying the condition p
sBoolean removeTasks(Predicate<Task> p);	Remove the tasks satisfying the condition p

In particular, the method *removeTasks* returns true if there exists a task object removed, and returns false otherwise. To support the flexible construction of the predicates, you should implement the following methods to construct an atomic predicate and their logical connections:

Method	Method Description
Predicate<Task> byType(TaskType type);	Return an atomic predicate determining whether a task object satisfying the conditions in 3(a)~3(e) in Section 2.
Predicate<Task> byTag(String tag);	
Predicate<Task> byTitle(String keyword);	
Predicate<Task> byDescription(String keyword);	
Predicate<Task> byCreationTime(LocalDate d);	
List<Predicate<Task>> genPredicates(Function<String, Predicate> f, List<String> strs);	Generate a list of predicates
Predicate<Task> andAll(List<Predicate<Task>> ps);	Return a predicate which is a logical connection of a list of atomic predicates.
Predicate<Task> orAll(List<Predicate<Task>> ps);	
Predicate<Task> not(Predicate<Task> p);	

Particularly, the method *genPredicates* receives a function mapping a string to a predicate over task objects and a list of strings, and returns a list of the predicates. In the following code snippet, for example, the list object *pls* contains five predicates examining whether a task object has one of the five words as its tag.

```
List<String> keywords = Arrays.asList("Java", "Lecture", "Lab", "Demo", "Assignment");
List pls = genPredicates(byTag, keywords);
```

The returned predicate of the method *andAll* holds for a task object if and only if all the predicates in the parameter *ps* hold. Similarly, the returned predicate of the method *orAll* holds for a task object if and only if at least one predicate in the parameter *ps* holds. The returned predicate of the method *not* holds as long as the parameter *p* does not hold.

We do not have any restriction on the complexity of your implementation. When testing your program, we will generate a series of the task objects and invoke the function *appendNewTask* multiple times to saturate the system. Then we will invoke the above methods randomly and compare the returned values with the correct ones. Thus, PLEASE pay more attention to the methods that modifying the task list in the system, as any wrong implementations of these methods would yield wrong returns of other methods.

4. Submission Details

The assignment will be due on: 23:59, 8 May, 2022. You are required to submit all your source codes through a zip file. You should compress the directory *Pokemon_PA3* to a .zip file and then submit the zip file to Canvas.