2590. Design a Todo List

Description

Design a Todo List Where users can add **tasks**, mark them as **complete**, or get a list of pending tasks. Users can also add **tags** to tasks and can filter the tasks by certain tags.

Implement the `TodoList` class:

- `TodoList()` Initializes the object.
- `int addTask(int userId, String taskDescription, int dueDate, List<String> tags)` Adds a task for the user with the ID `userId` with a due date equal to `dueDate` and a list of tags attached to the task. The return value is the ID of the task. This ID starts at `1` and is **sequentially** increasing. That is, the first task's id should be `1`, the second task's id should be `2`, and so on.
- `List<String> getAllTasks(int userId)` Returns a list of all the tasks not marked as complete for the user with ID `userId`, ordered by the due date. You should return an empty list if the user has no uncompleted tasks.
- `List<String> getTasksForTag(int userId, String tag)` Returns a list of all the tasks that are not marked as complete for the user with the ID `userId` and have `tag` as one of their tags, ordered by their due date. Return an empty list if no such task exists.
- `void completeTask(int userId, int taskId)` Marks the task with the ID `taskId` as completed only if the task exists and the user with the ID `userId` has this task, and it is uncompleted.

Example 1:

```
Input
["TodoList", "addTask", "addTask", "getAllTasks", "getAllTasks", "addTask", "getTasksForTag", "completeTask", "completeTask", "getTasksForTag",
"getAllTasks"]
[[], [1, "Task1", 50, []], [1, "Task2", 100, ["P1"]], [1], [5], [1, "Task3", 30, ["P1"]], [1, "P1"], [5, 1], [1, 2], [1, "P1"], [1]]
Output
[null, 1, 2, ["Task1", "Task2"], [], 3, ["Task3", "Task2"], null, null, ["Task3"], ["Task3", "Task1"]]

Explanation
TodoList todoList = new TodoList();
todoList.addTask(1, "Task1", 50, []); // return 1. This adds a new task for the user with id 1.
todoList.addTask(1, "Task2", 100, ["P1"]); // return 2. This adds another task for the user with id 1.
todoList.getAllTasks(1); // return ["Task1", "Task2"]. User 1 has two uncompleted tasks so far.
todoList.getAllTasks(5); // return []. User 5 does not have any tasks so far.
todoList.addTask(1, "Task3", 30, ["P1"]); // return 3. This adds another task for the user with id 1.
todoList.getTasksForTag(1, "P1"); // return ["Task3", "Task2"]. This returns the uncompleted tasks that have the tag "P1" for the user with id 1.
todoList.completeTask(5, 1); // This does nothing, since task 1 does not belong to user 5.
todoList.completeTask(1, 2); // This marks task 2 as completed.
todoList.getTasksForTag(1, "P1"); // return ["Task3"]. This returns the uncompleted tasks that have the tag "P1" for the user with id 1.
                                  // Notice that we did not include "Task2" because it is completed now.
todoList.getAllTasks(1); // return ["Task3", "Task1"]. User 1 now has 2 uncompleted tasks.
```

Constraints:

- `1 <= userId, taskId, dueDate <= 100`
- `0 <= tags.length <= 100`
- `1 <= taskDescription.length <= 50`
- `1 <= tags[i].length, tag.length <= 20`
- All `dueDate` values are unique.
- All the strings consist of lowercase and uppercase English letters and digits.
- At most `100` calls will be made for each method.