Activity 6.4 - Writing Classes II (ToDo List)

Overview

In this activity, you will create an *aggregate* class, `ToDoList`, which *has-a* list of `Task` references. Your `ToDoList` class will provide methods for managing tasks.

Instructions

Getting Started

1. Create a new folder "Activity6.4" in VS code and copy your Task.java, TaskMaster.java, and Category.java from Activity 6.3 to this project directory.
2. Import ToDoListInterface.java.
3. Create a new class named `ToDoList`.

`Task` Updates

Update the class header for your `Task` class to implement the `Comparable<Task>` interface:

```
public class Task implements Comparable<Task>
```

Add equals() and compareTo() methods to your Task class

```
/**
 * Indicates whether some task is "equal to" this task. Tasks are equal if
 * they have the same description (case insensitive).
 * @param The other task to compare this task to.
 * @return true if the tasks are equal, false otherwise.
 */
public boolean equals(Task other)
{
	if(this.description.equalsIgnoreCase(other.description)
			&& this.category.equals(other.category))
	{
		return true;
	}
```

```
        else
        {
                return false;
        }
}
/**
 * Compares two tasks based on their priorities.
 * @param The other task to compare this task to.
 * @return difference between the tasks if they are either both complete or
 * both incomplete, return -1 if this task is complete and the parameter task
 * is incomplete, returns 1 if parameter task is complete and this task is not
 * complete.
*/
@Override
public int compareTo(Task other)
{
        if(this.complete == other.complete)
        {
                return this.priority - other.priority;
        }
        else if(this.complete && !other.complete)
        {
                return -1;
        }
        else
        {
                return 1;
        }
}
```

`ToDoList` Requirements

Implement the ToDoListInterface

Your ToDoList class must implement the provided ToDoListInterface. Modify the class header as follows.

```
public class ToDoList implements ToDoListInterface
```

Instance Variables

Every ToDoList will have the following instance variables (aka attributes).

- A *`name`*.
- An `ArrayList<Task>` of *`tasks`*.

Constructors

The constructor for a `ToDoList` should take the name of the list as a parameter and instantiate the `ArrayList<Task>` of *`tasks`*.

toString Method

Write a `toString` method that will return a String containing the name and tasks in the `ToDoList`. Use the following format.

```
-------------
My ToDo List
-------------
[X] Finish Activity 14, 0, NONE
[ ] Give Tigger a bath, 20, NONE
```

Methods

Your `ToDoList` must implement the following methods.

- `getName()` - Returns the name of the ToDoList.
- `addTask(Task t)` - Adds the given task to the ToDoList if a duplicate task is not already in the list. **Hint**: We wrote an .equals() method in Task which is used by the .contains() method in the ArrayList class.
- `addTask(String description)` - Creates and adds a new `Task` with the given description to the `ToDoList` if a duplicate task is not already in the list.
- `getWork()` - Returns the next incomplete task with the highest priority. Returns `null` if there are no tasks in the list or if all the tasks are complete.
  - **Hint**: The `isEmpty()` method from the `ArrayList` class will tell you if your list has any tasks in it.
  - To find the next incomplete task with the highest priority, you will leverage the Java Collections API and the `compareTo` method that you added to your `Task` class in the previous section of this activity.

    The Java Collections API provides a `Collections` class which consists exclusively of static methods that operate on or return collections. Since an `ArrayList` is a type of Collection, we can use these methods to help manage our task list.

    Use the max method from the Collections class to get the greatest task in the list. The max method returns the maximum element of the given collection, according to the natural ordering of its elements. The "*natural ordering*" for a `Task` has already been defined by your `compareTo` method.

```
Task maxTask = Collections.max(tasks);
```

  - Make sure to check if the `maxTask` is complete before returning it to the user. If the `maxTask` is complete (and you implemented your compareTo correctly), you can assume there is no work to do and return `null`.

- `getTaskList()` - Returns a **copy** of the `ArrayList` of tasks. (Hint: not an alias.)

`TaskMaster` Requirements

In your `TaskMaster` "driver" class, you will test the constructors and methods of your `ToDoList` class by creating a `ToDoList` instance.

- Create a new `ToDoList` instance using your constructor.
- Add `task1`, `task2`, and `task3` to your list using your `addTask(Task t)` method.
- Add another task to your list using your `addTask(String description)` method.
- Print your `ToDoList` and verify that your `toString` works as expected.
- Call the `getWork` method on your list to get your next task. Did it return the correct task?
- Test the following conditions on your getWork method. How will you set up a list to force the conditions?
  - Does it return null if your list is empty?
  - Does it return null if all tasks are complete?

Terminology Identification

In your code add comments identifying examples of the following: aggregation, dependency, inheritance, encapsulation. These should be identified with an @keyterm tag within the comment.

Code Review

When you are finished with this activity, pair up with a classmate and review each other's code to make sure it meets all the requirements.

Submission

After completing the assignment, use the assignment link in Canvas and follow the submission instructions there. You will upload your .java files and put your reflection in the "Comments" box.

Reflection Requirements

Write a one paragraph reflection describing your experience with this activity. The reflection should also include the name of your code review partner AND something interesting you found in their code. Please review the activity rubric for more details.