Uttara InfoSolutions

www.uttarainfo.com

Prototype and steps to implement ToDoApp Project:

When the ToDoApp solution is run, the following should be displayed:

```
Press 1 to Create Category
Press 2 to Load Category
Press 3 to Search
Press 4 to List
Press 5 to Exit
Enter choice:
```

If 1 is given as input,
a) you should first ask the user to enter name of category.
You should validate whether this name is already used by another as a business validation (assume the path to a folder and concat the name to this and using File class, validate).
b) Then you should display the following menu:

```
Press 1 to Add a Task
Press 2 to Edit a Task
Press 3 to Remove a Task
Press 4 to List the Tasks
Press 5 to Search
Press 6 to Go back
```

If 1 is selected,
i) you should ask the user to input a task name (this has to be unique)
ii) then take inputs of description, due date, priority, tags (comma separated words)

Implementation tip: Once all the details of the task is given, you should insert the task info into the file. Concat the info like this:

```
taskname:description:status:priority:cr_dt:due_dt:tag1,tag2,tag3..tagx
```

vi) When 2 is selected (Edit), you should allow the user to a) remove or change individual task info

vii) A user can remove a task that exits only. All its info has to be removed from the file.
viii) When 4 is selected (List), the user should be shown the following submenu:

Press 1 to list tasks by alphabetical listing by name
Press 2 to list tasks by due date
Press 3 to list tasks by created date
Press 4 to list tasks by longest time

Depending on user input, the sorting of tasks have to be done.

ix) When 5 is selected (Search), you should ask the user to input a string to search. This string has be searched in the entire contents of the categories(name, email,phone num, tags, etc) and the following should be displayed:

```
Total number of occurances : <num>
Number of occurances in description: <num>
Matches found:
<task name1 - desc text1>
<task name2 - desc text2>
...
Number of occurances in name: <num>
<task name1>
<task name2>
...
Number of occurrances in tags: <num>
<task name ->
<task name ->
```

In the outer menu, if 2 is given as input (Load Category), you should ask the user to input the name of category (which should exist) and then you will display the same inner menu where in the user can now add,edit,remove,list,search from the loaded tasks.

Implementation tips:
a)Implement the TodoApp solution using basic MVC demoed example. Simply create a class with main() method that interacts with user and provides functionality. No need to create any other class. Create static methods to reuse validation.

Classes to create:
StartApp with main() will act as the View to the user. In the main(), code the menu, user interaction, taking input, showing messages, invoking business logic and handle exceptions.

A TaskBean class to hold data about a task (all has-a relationship create as ins. var).

TasKBean class represents the data holder:
- name
- desc
- cr_dt
- end_dt
- priority
- status
- tags

TodoModel will hold business methods like:
public String addTask(TaskBean t,String catName)
public List<TaskBean> search(String str)
public String checkIfCategoryExists(String name)
public String update(TaskBean old, TaskBean new)
public String delete(String taskName)
public List<TaskBean> getTasksBasedOnCrDate(String catName) // for category
public List<TaskBean> getTasksBasedOnCrDate() // for all categories
public List<TaskBean> getTasksBasedOnDueDate(String catName)
public List<TaskBean> getTasksBasedOnDueDate()
public List<TaskBean> getTasksBasedOnTimePending(String catName)
public List<TaskBean> getTasksBasedOnTimePending()

<u>Steps to build the App:</u>

1) Implement the Menu in the StartApp main() first. You should be able to go through the entire app`s menu using the options. This will help understand the control.
2) First implement the addTask() use case without bothering about implementing any validations. You will have to create a TodoModel and TaskBean class. The bean is used only to hold data so that it can be passed as a parameter between the ViewController & the Model. The main() which acts as view is the only method that should perform SOP, accept i/p from user. Other methods should return messages or throw exceptions that the main() should catch and display to user. Understand the control flow and data flow.

StartApp->main() invokes model.addTask(bean,catName) and addTask() returns String to main() to indicate success/failure. If not success, the returned string represents the error message to be displayed to the user.

```
main()
{
        …
        case XX:
                    …
SOP("Enter category name");
catName = sc2.nextLine();
….
….
case YY:
….
// read all task info and create TaskBean object and inject the
data into it by using parameterised constructor.
String msg = model.addTask(bean,catName);
if(msg.equals("success"))
// indicates addition succeeded, show msg to user
else
SOP(msg); // display error msg to user
}
```

So StartApp->main()->display menu, accept category name as input, invoke model.checkIfCategoryExists(), if it does not, then create Task bean object with all user inputs and pass to model.addTask() method.

model.checkIfCategoryExists():
When a category name as input, create a java.io.File instance with the input name as constructor argument, verify if this file exists and if not, you can create the file. It is into this file that you will write the contents once the user adds/edits/removes a task.Use a FileWriter encapsulated in a BufferedWriter to do file writing (Pass true as its second parameter to append to the file). In the file, one line represents one tasks input. You can use BufferedWriters newLine() method to insert a new line in the file.

3) Implement the listTasks() use case:
Again understand the control and data flow. The tasks data is in the file named by the category name. Who can communicate to the file system? model. Who can invoke the model? the ViewController main(). So when the user chooses the list option, invoke model.listTasks(catName) and pass category name as parameter. The method is going to return List<TaskBean> which contains all the tasks data in the file as beans. The main() will loop over the collection, get each bean, invoke getter methods and display to user.

4) Implement the search use case:
Same as the list use case but instead pass the user search string as input to the search(srchString, catName) method of model. The method will return a List<TaskBean> which contains the searched string in its data. The model will read each line from the category file, and check if the line contains the search string and

if yes, will split it and inject into a bean which is then put into a list. After going through all the lines, it will return the list to main(). main() will loop over and display the list to user.

5) Implement sorting:
Same as list. Build Comparators to implement sorting TaskBean by name, date, etc and pass to Collections.sort(list,comparator ref) to sort the data before displaying to user.

6) Implement delete usecase:
main() will invoke model.deleteTask(taskName,catName) and pass task name as parameter to it. The model will open the category file, read each line into a collection except if it contains the taskName. Then create a new FileWriter(catName) without the appending parameter, loop over the collection and write the contents to the file. This will overwrite the same file with all the same contents except the line to be deleted.

7) Implement edit usecase:
Same as delete. First take the task name as input from the user, ask model to return the TaskBean for this name that exists. Show the content to user and display this menu:
Press 1 to change description
Press 2 to change status
Press 3 to change end date
etc

Now take the input from user, create a bean with the new data, pass this to the model to update. The model method should follow the same steps as delete but add the edited task line to the collection before writing it back to the file.

<u>Things to take care:</u>

1) try..catch exceptions in main() so that the app does not crash
2) All communication from main() to model and vice versa should involve passing of beans, Strings or primitives and max number of parameters should be 3-4 for any method. The model should return collections/bean if many values or many beans must be returned
3) All i/o should be done in main(), all file i/o should be done in model methods
4) Bean should override equals(), hashCode(), toString(), must be package, expose setter/getters, implement natural ordering by implementing Comparable interface
5) You have to perform validations for all use cases - both input and business and test all cases
6) Please follow naming conventions!! All class names should be starting with capital letter and co-joined words must be title case. All variables and method

names should start with a small case and co-joined words must be title case. Names of methods must be a verb and must indicate that it does correctly. Name of variables should indicate what data it holds correctly.

You can implement and complete this project in 3-5 days time. Take the assistance of Lab Instructors / mine when required. Please do this individually and not a group as there will be many learnings and I do not want you to miss out on this. Thanks for the effort! Happy coding!