C0

TodoListManager

We've given you the method signatures as scaffolding for this assignment. This should also help you develop your program in stages. More specifically, **we've broken down the assignment into four parts** in which you can think about your program.

- In the first part of your program, you will write the overall structure of the program and a method that prints your TODOs.
- In the second part, you will allow a user to add and remove items from the TODO list.
- In the third part, you will add the ability to save and load TODOs from a file.
- In the fourth part, you will add your creative extension to the program.

A sample execution of the completed program can be found below.

We suggest that after completion of each part, you submit to run tests. There will be tests corresponding to each part. After parts 1 and 2, you will still have some failing tests, but all tests should pass after the completion of all three parts.

Part 1 - Initial Setup

For this first part, you will implement the scaffold for the **main** method and the **printTodos** method.

- The main method should **declare an ArrayList** of TODOs, which it can pass to other methods and should have a **loop** that continues to prompt the user for an action. We recommend writing out the whole user interaction loop even if you haven't implemented the methods for future parts, such as adding to the TODO list.

  The user's input for an action should be processed **case-insensitively**. When user input is unrecognized, print `Unknown input:` followed by the user's input. Your loop should continue to re-prompt the user for an action until the user types `Q` or `q`.

  Hint: You'll find the method `equalsIgnoreCase` helpful for comparing strings case-insensitively.

- The **printTodos** method takes a List of TODOs and prints out each item in the todo list, numbered. Your method should also print a header for the TODO list, as `Today's TODOs:` . When the list is empty, print `You have nothing to do yet today! Relax!`.

With these two tasks completed, you should be able to run your program and get the following output.

Welcome to your TODO List Manager!
What would you like to do?
(A)dd TODO, (M)ark TODO as done, (L)oad TODOs, (S)ave TODOs, (Q)uit?
F
Unknown input: F
Today's TODOs:
  You have nothing to do yet today! Relax!

What would you like to do?
(A)dd TODO, (M)ark TODO as done, (L)oad TODOs, (S)ave TODOs, (Q)uit?
q

Part 2 - Adding and Removing TODOs

For the second part, implement the **addItem** and **markItemAsDone** methods.

- The **addItem** method takes a console `Scanner` and a `List` of TODO items to modify. The `addItem` method should allow a user to add a TODO to their TODO list.

  Each time the user adds an item, except when the list is empty, they are asked *where* the new TODO should be inserted. The user can also type in the number between 1 and one greater than the last TODO item to manually insert at the end.

  Additionally, the user can hit enter without typing a number when prompted for a number to automatically insert at the end. In the sample program execution, when the user wanted to add "Study for BIO 180", the user hit "enter" to indicate the item should be added to the end of the list. (The enter cannot be explicitly seen in the sample execution).

  See the note below on assumptions you can make about user input. There is also a summary of this behavior in the Development Notes below.

- The **markItemAsDone** method takes a console Scanner and a `List` of TODO items to modify.

  The `markItemAsDone` method should allow a user to mark a TODO as completed, removing it from the TODO list. When prompted for an item to remove, the user should indicate which item they wish to remove using the item's number.

  If the TODO list is empty, it should not prompt the user for a tasks number and instead should print out a message: `All done! Nothing left to mark as done!`

  Notice in the sample program execution when there is one item is the list left to remove, the prompt will `Which item did you complete (1-1)?`. This is expected.

  See the note below on assumptions you can make about user input.

Development Notes

Assumptions About User Input

For `addItem` and `markItemAsDone`, you may assume that if the user is prompted to type in a number for insertion/removal, that they type in a valid number within the prompted range. The one exception to this is in `addItem`, the user can type in nothing (thus returning an empty String `""`).

You do not need to handle the other cases if the user types in something that is not a number there or if they type a number outside the valid range. Pay particular attention to the example below to see what the valid ranges should be and to verify that hitting enter is valid in the `addItem` case.

- You should read ***all*** user input from your console Scanner using line-based method `.nextLine()` . Using both token-based and line-based processing with the same Scanner object can lead to odd behavior and bugs so you should avoid calling anything but `.nextLine()` for this assignment.

- Since you should read all user input using `.nextLine()`, you can not read integer input with `.nextInt()` . When prompting for integers, read user input as a `String` with `.nextLine()` and convert it to an `int` in the following way:

```
String input = console.nextLine();
int num = Integer.parseInt(input);
```

You may assume valid user input, as your program will crash if the user does not type a valid integer.

- When the user provides numerical input to indicate insertion or removal of an item, they will use the 1-based numbering printed in the user interface. Be careful to add to and remove from the correct corresponding position in your ArrayList!
- Since the description for `addItem` is a little tricky, we wanted to explicitly highlight the cases you need to consider:
  - When the list of items is empty.
  - When the list of items is non-empty, and the user hits enter right away when prompted for where to insert the TODO.
  - When the list of items is non-empty, and the user types in a number when prompted for where to insert the TODO.

At this point, you should be able to run the program and generate the following log of user interaction:

```
Welcome to your TODO List Manager!
What would you like to do?
(A)dd TODO, (M)ark TODO as done, (L)oad TODOs, (S)ave TODOs, (Q)uit? A
What would you like to add? Workout
Today's TODOs:
  1: Workout
What would you like to do?
(A)dd TODO, (M)ark TODO as done, (L)oad TODOs, (S)ave TODOs, (Q)uit? a
What would you like to add? Study for BIO 180
Where in the list should it be (1-2)? (Enter for end):
Today's TODOs:
  1: Workout
  2: Study for BIO 180
What would you like to do?
(A)dd TODO, (M)ark TODO as done, (L)oad TODOs, (S)ave TODOs, (Q)uit? A
What would you like to add? Complete 122 Creative Project 0
Where in the list should it be (1-3)? (Enter for end): 2
Today's TODOs:
  1: Workout
  2: Complete 122 Creative Project 0
  3: Study for BIO 180
What would you like to do?
```

```
(A)dd TODO, (M)ark TODO as done, (L)oad TODOs, (S)ave TODOs, (Q)uit? mark
Unknown input: mark
Today's TODOs:
  1: Workout
  2: Complete 122 Creative Project 0
  3: Study for BIO 180
What would you like to do?
(A)dd TODO, (M)ark TODO as done, (L)oad TODOs, (S)ave TODOs, (Q)uit? M
Which item did you complete (1-3)? 3
Today's TODOs:
  1: Workout
  2: Complete 122 Creative Project 0
What would you like to do?
(A)dd TODO, (M)ark TODO as done, (L)oad TODOs, (S)ave TODOs, (Q)uit? Q
```

Part 3 - Saving and Loading TODOs from a File

For the last part, implement the **saveItems** and **loadItems** methods.

- The **saveItems** method takes a console `Scanner` and a `List` of TODO items to be saved to a file. This method should prompt the user for a file name and print each TODO list item to the file, in order, one item per line.
- The **loadItems** method takes a console `Scanner` and a `List` of TODO items to modify. This method should prompt the user for a file name containing the list of TODO items, with each item on one line. The file should be read, line by line, with each item being added, in order, to the TODO list. If the TODO list previously contained items, its contents should be completely replaced by the contents of the file.

Development Notes

- If you would like a refresher on File I/O, or how to do File I/O in Java, you may find the Java Tutorial for File I/O helpful. Also check out the Pre-Class Material from Friday about how to write to files!

Assumptions About User Input

For the `loadItems` method, you can assume the user will type in the name of a file that exists. You should make no assumptions about the number of tasks in the file. However, you still need to write the method header to say `throws FileNotFoundException` since Java requires that for reading from files.

After completing this part, you should be able to recreate the whole user interaction from earlier.

Sample Program Execution (user input underlined/bold)

```
Welcome to your TODO List Manager!
What would you like to do?
(A)dd TODO, (M)ark TODO as done, (L)oad TODOs, (S)ave TODOs, (Q)uit? A
What would you like to add? Workout
```

```
Today's TODOs:
  1: Workout
What would you like to do?
(A)dd TODO, (M)ark TODO as done, (L)oad TODOs, (S)ave TODOs, (Q)uit? a
What would you like to add? Study for BIO 180
Where in the list should it be (1-2)? (Enter for end):
Today's TODOs:
  1: Workout
  2: Study for BIO 180
What would you like to do?
(A)dd TODO, (M)ark TODO as done, (L)oad TODOs, (S)ave TODOs, (Q)uit? A
What would you like to add? Complete 122 Creative Project 0
Where in the list should it be (1-3)? (Enter for end): 2
Today's TODOs:
  1: Workout
  2: Complete 122 Creative Project 0
  3: Study for BIO 180
What would you like to do?
(A)dd TODO, (M)ark TODO as done, (L)oad TODOs, (S)ave TODOs, (Q)uit? mark
Unknown input: mark
Today's TODOs:
  1: Workout
  2: Complete 122 Creative Project 0
  3: Study for BIO 180
What would you like to do?
(A)dd TODO, (M)ark TODO as done, (L)oad TODOs, (S)ave TODOs, (Q)uit? M
Which item did you complete (1-3)? 3
Today's TODOs:
  1: Workout
  2: Complete 122 Creative Project 0
What would you like to do?
(A)dd TODO, (M)ark TODO as done, (L)oad TODOs, (S)ave TODOs, (Q)uit? S
File name? todo.txt
Today's TODOs:
  1: Workout
  2: Complete 122 Creative Project 0
What would you like to do?
(A)dd TODO, (M)ark TODO as done, (L)oad TODOs, (S)ave TODOs, (Q)uit? m
Which item did you complete (1-2)? 1
Today's TODOs:
  1: Complete 122 Creative Project 0
What would you like to do?
(A)dd TODO, (M)ark TODO as done, (L)oad TODOs, (S)ave TODOs, (Q)uit? M
Which item did you complete (1-1)? 1
Today's TODOs:
```

```
  You have nothing to do yet today! Relax!
What would you like to do?
(A)dd TODO, (M)ark TODO as done, (L)oad TODOs, (S)ave TODOs, (Q)uit? L
File name? todo.txt
Today's TODOs:
  1: Workout
  2: Complete 122 Creative Project 0
What would you like to do?
(A)dd TODO, (M)ark TODO as done, (L)oad TODOs, (S)ave TODOs, (Q)uit? q
```

Part 4 - Creative Extension

This is where you should get creative! Add a new feature to your TodoList manager. As this is meant to be a creative portion where you can choose to implement what you want, there are not many hard requirements on what your extension should do. The requirements we do have are as follows:

- When the `EXTENSION_FLAG` is `false`, your extension should be off and your program should *exactly* produce the output described in the structured portion above. This means you should pass all of the tests when pressing "**Test**".
- Your extension should be a meaningful extension to the application to add some new feature or change the behavior of the TODO list in a significant way. As a baseline, the new feature should allow the user to change the state of the application in some novel way.
    - If you need to create additional ArrayLists to make your extension work, that is allowed! Just remember to not create them or use these extra lists if the feature flag is off.
- Note that any extension that is meaningful and changes the behavior of the TODO list in a significant way will likely require at least two ArrayList methods calls! The ArrayList Cheatsheet includes all of the ArrayList methods we've learned.
- The functionality of the TODO list (adding TODOs, removing TODOs) should still exist when your extension is enabled, but it is okay to modify those original behaviors as part of your extension. For example, one of the example extensions we suggested changes how the user can add multiple items at a time.

While you may add any extension you would like, here are some possible ideas to get you started

- Allow a user to add multiple items at a time (instead of having to type 'A' each time they wish to add an item)
- Display the list of completed TODOs
- Allow the user to reorder TODO items (designing a text-based UI may be tricky!)
- Allow the user to remove items from the list using the name of the item
- Allow the user to specify a deadline associated with each item

Use your imagination to try to implement something you are interested in! Do note that it will help to plan out how you might have the change your solution to work with your extension, while still meeting the requirement

that your extension should be disabled if the feature flag is `false`.

Additionally, in the reflection for this assignment, you will explain what extension you implemented and how it is supposed to behave. Part of your grade for the Behavior component will come from the grader trying out the feature as you described and making sure it behaves as you described.

Make sure to test out your feature to make sure it works before finishing your submission!

Assignment Requirements

For this assignment, you should follow the Code Quality guide when writing your code to ensure it is readable and maintainable. In particular, you should focus on the following requirements:

- For this part of the assignment, you should only be using a single `ArrayList` to manage the TODO list Outside of the one you are supposed to create in `main`, you should not construct any additional `ArrayList` instances when your extension feature flag is `false`.
    - For the creative portion, you're more than welcome to use additional `ArrayList`s for your extension!
- You should use the `List` interface type where appropriate and properly use type parameters when specifying the types of our `ArrayList`.
- You should comment your code following Commenting Guide. You should write a header comment, a class comment, and a comment for every method other than main.