Assignment 5 (Part 2)

- Due Wednesday by 20:59
- Points 0
- Available after 14 Mar at 18:00

1 Introduction

In this assignment, we will extend the functionality of the calendar application with additional features. Hopefully your current design of the calendar application is flexible and scalable in a way that will allow you to incorporate the additional requirements with minimal changes to the existing implementation.

2 Support multiple calendars

The previous iteration of the calendar application allowed users to create, modify, and query events for a single calendar. You must now extend the capability of your application to support multiple calendars.

The application should support the ability to create and maintain several calendars, each with its own set of events and a unique name. The calendar name will be used to identify the calendar. It should be possible to edit the name of a calendar. Two calendars cannot have the same name.

3 Support Timezones

Currently the times specified for events are assumed to be in reference to a default timezone (presumably EST). You must now support the ability to manage different timezones.

Specifically, each calendar is associated with a specific timezone. All events in that calendar are assumed to be in its timezone (and thus, timezone does not have to be specified when dealing with events). It should be possible to change the timezone of a calendar.

4 A Clearer Command Set

When the calendar application was released to some users to get feedback, many users reported confusion about conflicts, editing events and how they could have cascading effects in other events. Therefore, the following updated features were requested, which you must now support:

- Events in a calendar cannot conflict with each other. This means that declining creation of new events when they conflict with existing events is the default and only behavior. A conflict with any instance of a recurring event is treated as a conflict with the recurring event itself, and is prohibited.
- Editing an existing event that can create a conflict with another existing event is not allowed.

Existing commands that allow the `--autoDecline` flag must now treat this as the default rule (i.e. conflicts are declined whether or not this flag is specified).

- `edit events <property> <eventName> from <dateStringTtimeString> with <NewPropertyValue>`: this command should change the specified property to the new value for **all** events that have the given name and that start at or after the given date and time. The clarification is that the "from" does not imply an event should start at the given date and time, but any time at or after.

5 Copy events

Several users also suggested that given the application can support multiple calendars, they should be able to copy events to another timeline, even across calendars. For example, the order of lecture topics in CS 5010 stay the same from one semester to another, so these events could simply be copied *en-masse* to a time line that starts with a specified date (for the next semester). For example, *copy all events for lecture topics from September 5 2024 to December 18 2024 (Fall 2024 semester) to a place in the calendar that starts at January 8 2025 (start of the Spring 2025 semester).

Based on this need you must add support to copy a single or recurring event from a specified (source) calendar to a specified (starting) date and time in a specified (target) calendar. You must also support copy all events within a specified interval to a specified (starting) date and time. In both cases, the source and target calendars may or may not be the same.

Copying an event to a target calendar must be considered a new event in the target calendar, and therefore conflicts must be managed consistent with the above.

6 Support for additional commands

The following commands must be supported in the text-based interface:

`create calendar --name <calName> --timezone area/location`

This command will create a new calendar with a unique name and timezone as specified by the user. The expected timezone format is the **IANA Time Zone Database format (https://en.wikipedia.org/wiki/List_of_tz_database_time_zones)**. In this format the timezone is specified as "area/location". Few examples include "America/New_York", "Europe/Paris", "Asia/Kolkata", "Australia/Sydney", "Africa/Cairo", etc. The command is invalid if the user provides a non-unique calendar name or an unsupported timezone.

`edit calendar --name <name-of-calendar> --property <property-name> <new-property-value>`

This command is used to change/modify an existing property (`name` or `timezone`) of the calendar. The command is invalid if the property being changed is absent or the value is invalid in the context of the property.

`use calendar --name <name-of-calendar>`

A user can create/edit/print/export events in the context of a calendar. They can use this command to set the calendar context. Note this means that the commands in the previous iteration only make sense when a calendar is in use (i.e. some calendar must be in use for them to work). Otherwise, they are invalid.

`copy event <eventName> on <dateStringTtimeString> --target <calendarName> to <dateStringTtimeString>`

The command is used to copy a specific event with the given name and start date/time from the current calendar to the target calendar to start at the specified date/time. The "to" date/time is assumed to be specified in the timezone of the target calendar. The copied event must be in the same timezone as the target calendar.

```
copy events on <dateString> --target <calendarName> to <dateString>
```

This command has the same behavior as the `copy event` above, except it copies all events scheduled on that day. The times remain the same, except they are converted to the timezone of the target calendar.

```
copy events between <dateString> and <dateString> --target <calendarName> to <dateString>
```

The command has the same behavior as the other copy commands, except it copies all events scheduled in the specified date interval. The date string in the target calendar corresponds to the start of the interval. The endpoint dates of the interval are inclusive.

7 Miscellaneous Requirements and Recommendations

7.1 Design Considerations

As you add these features, pay attention to your design.

- What changes in your existing design do you make, and why?
- What is the best way to incorporate new features into an existing application? Do you support all features, or do you release incremental versions with different features (e.g. Starter Edition, Pro Edition, etc.)?
- As you make changes, are you still adhering to the MVC architecture? Are you putting each class and operation where it belongs?
- Have your design enhancements gone beyond the specific operations to be implemented? How easy would it be to add similar extensions in the future?
- Whatever design choices you make, what are its advantages and limitations?

While you are allowed to change your design, you should be able to justify it. The bigger the change, the better we expect your justification to be. Please document and justify each change in a README file (maintain this file as you complete this assignment, rather than summarize after you are done and risk missing something).

7.2 Testing

You are expected to test each part of the program that you have written. Accordingly, you are expected to test your model, controller and if applicable, your view. Use of mocks to test the controller is highly recommended, but not required.

You are not allowed to use external tools (i.e. classes not within the JDK), with the exception of the JUnit testing framework.

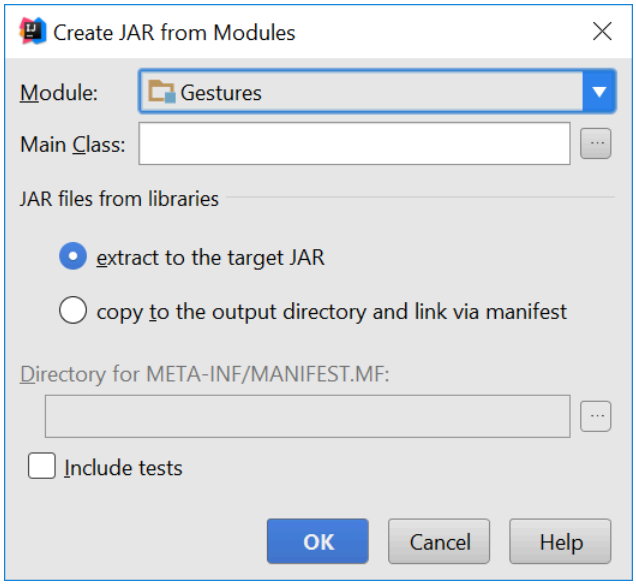
7.2.2 Testing Quality

You must evaluate the quality of your tests using the PIT mutation testing tool similar to how you have been using it for the previous assignments. Your tests will be graded based on both test strength and mutation coverage. Both coverages must be 100% for full credit for this part of the rubric.

7.3 Create a JAR file of your program

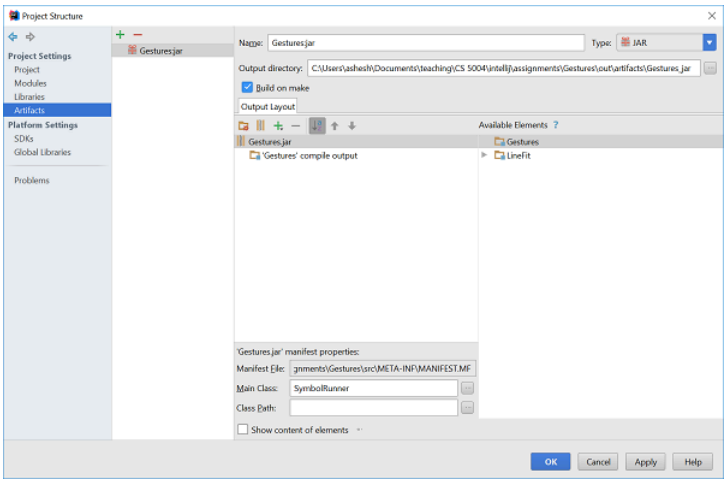
To create a JAR file, do the following:

- Go to File -> Project Structure -> Project Settings -> Artifacts
- Click on the plus sign
- Choose JAR -> From Modules with dependencies. You should now see the module in your project that you are working on (may be different than what is shown in the image below)



- Remove any dependencies that are not necessary to run the program (e.g. PIT mutation tool)
- Select the main class of your program (where you defined the `main(String[] args)` method)

- If you see a checkbox labelled "Build on make", check it.
- Hit ok
- You should now see something like



If now you see a checkbox labelled "Build on make", check it now.

- Make your project (the button to the left of the run configurations dropdown, with the ones and zeros and a down-arrow on it). Your .jar file should now be in /out/artifacts/.
- **Verify that your jar file works** . To do this, copy the jar file to another folder. Now open a command-prompt/terminal and navigate to that folder. Now type java -jar NameOfJARFile.jar and press ENTER. The program should behave accordingly. If instead you get errors review the above procedure to create the JAR file correctly. **You can also run the jar file by double-clicking on it (it will run without command-line arguments).**

8 What to Submit

A complete submission must contain the following:

- All your code in the src/ folder
- All your tests in the test/ folder
- A res/ folder with the following:
  - A JAR file of your application
  - A screenshot of the Google Calendar with your events
  - A txt file illustrating all the valid commands that work in your program. We will begin grading by running the program in headless mode with this file
  - A txt file illustrating at least one invalid command.
  - A class diagram representing the structure of the application's design. The class diagram should show names of classes and interfaces, signature of methods and relationships (inheritance and composition). Do not show field names they may clutter the diagram. We expect these to be not hand-scribbled, you stand to lose points for submitting a picture of a hand-drawn diagram.
- A README.md file with the following information:

  a. A list of changes to the design of your program, along with a brief justification of each. **Describing changes only in paragraph form will result in a point deduction.**

  b. Instructions on how to run your program (using the jar file from the terminal).

  c. Which features work and which do not.

  d. A rough distribution of which team member contributed to which parts of the assignment.

  e. Anything else you need us to know when we grade.
- There is a file size limit of 7 MB on the server. Do not submit unnecessary files such as hidden folders or Java bytecode typically found in directories such as out/ and target/.

Grading standards

For this assignment, you will be graded on

- completeness of your submission, including all the expected examples
- adherence to an MVC design
- the design of your interface(s), in terms of clarity, flexibility, and how plausibly it will support needed functionality;
- the appropriateness of your representation choices for the data, and the adequacy of any documented class invariants (please comment on why you chose the representation you did in the code);
- the forward thinking in your design, in terms of its flexibility, use of abstraction, etc.
- the correctness and style of your implementation, and
- the comprehensiveness and correctness of your test coverage.