Phase 2 Design, Changes, and Features

Design Patterns:

A list of design patterns that you used and why/how. Please include the names of all classes involved in the implementation of the pattern.

- **Marker Interfaces:** to tag different managers and presenters as specific types to greatly reduce the number of parameters in constructor calls (Using an ArrayList of those types instead of having each individual usecase/presenter as a parameter)
    - IManager (implemented by EventManager, UserManager, SpeakerManager, PresenterManager)
    - IPresenter (implemented by RequestUI, MessageUI, EventUI, ErrorUI, LoginUI)
- **Dependency Injection:** Instead of creating new use cases/presenter in every controller, we passed in the different objects that were needed instead of initializing new ones each time, this helped us avoid hard dependency in the classes.
    - UserController (Injects IManager, IPresenter)
    - LoginController (Injects IManager, IPresenter)
    - EventController (Injects IManager, IPresenter)
    - MessageController (Injects IManager, IPresenter)
- **Builder:** Within our Main class we have a method that is dedicated to building the entire program such that in psvm we only have to call one method to run the program. Similarly in our LoginController we have a run function that will build and run one instance of our program for a single User
    - Main (runConference() method)
    - LoginController (run() method)

Design Decisions and Changes

A list of design decisions and explanations about how your code has improved since Phase 1. (Includes reasons we opted to not use some patterns)

- We reduced the number of methods in certain classes as some seemed to make more sense separately.
- Removed all print statements outside of presenter classes to follow clean architecture.
- Changed some method names to be more descriptive/
- Removed entity dependency from presenter class (EventUI) as they should be interacting with the Use case (EventManager) instead
- Applied dependency injection to the EventManager and presenters as it is better to have them be passed through so the same objects are being worked with rather than reinitializing them everytime the controller is called
- Initially to create the **user request** functionality, we intended to use the *Observer* design pattern in order to update and notify the user and organizers when the request had been accepted, however we opted out of using this

because Java 8's implementation of *Observer* used a class and creating a subclass Request from Message would not be a viable option (since Java 8 does not have multi-inheritance). Without inheritance the Request class would be essentially just duplicated code from Message with a minimal amount of extra features, but duplicate code is bad design. Instead, we opted to use boolean instance variables which controlled a message status as a normal message, pending request, or accepted request which was modified by downstream use cases and subsequent controllers depending on the user's type.

- Our initial idea for storing different **room** properties was to have them stored within the event, however this was actually not feasible due to the large amount of inefficient extending and refactoring that would need to happen. Instead we opted to create a new room entity that stored specific information about the event and was easily extendable from our previous code and managed via EventManager with little code changes. This allowed for cleaner classes, more extensibility, and better design.
- Changed the data storing method from serializing variables in use case EventManger to directly serializing the whole EventManager, this allowed us to also store more variables in EventManager, as well as fix dependency issues in which the Gateway was directly interacting with entities.
- Fixed the way event IDs are created, instead of using eventList.size(), we created a variable that keeps track of the event counts, this allows us to delete an event entity completely from the event list when an organizer deletes an event and reduces the list size and memory usage.

Features

New additions:

- Created user requests, where when signing up for events as an Attendee a request can be sent. This request can be seen by all organizers and the request can be approved.
- New VIP events and users were created where only VIP users can join the VIP only events.
- Organizers are now able to create all types of users.
- Rooms and events are able to have set capacities where only a certain number of users may attend.
- Users are now able to output a pdf of the conference schedule.
- Event capacity introduced and can be changed by the organizer after creation.
- Multi-speaker events are available for creation that allows more than one speaker.
- Events can be deleted by organizers.
- Events have can be any duration within the available timeslots in terms of hours