Move to Change

Ana, Christian, Jimmy, Michael, and Owen

Overall Goals

To help activism-focused organizations with managing growth and membership at a growing scale

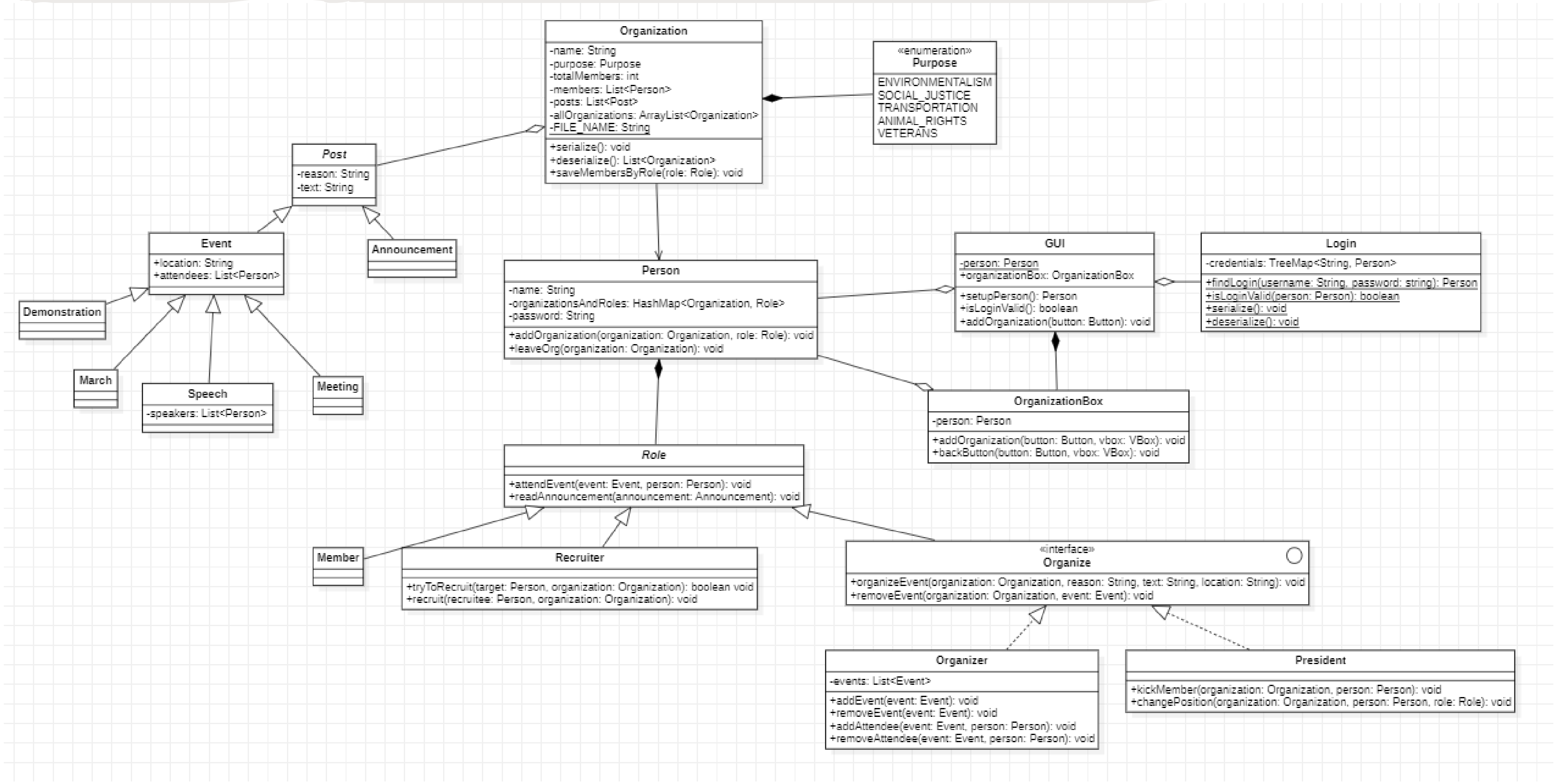
To assist with distributing content i.e., announcements and events

To help organizations or people with similar goals find each other

Why is this helpful?

Many activist organizations struggle to organize people and often, there are many organizations all separately trying to achieve a single goal

UML Class Diagram


Organization
-name: String
-purpose: Purpose
-totalMembers: int
-members: List<Person>
-posts: List<Post>
-allOrganizations: ArrayList<Organization>
-FILE_NAME: String
+serialize(): void
+deserialize(): List<Organization>
+saveMembersByRole(role: Role): void
«enumeration»
Purpose
ENVIRONMENTALISM
SOCIAL_JUSTICE
TRANSPORTATION
ANIMAL_RIGHTS
VETERANS
Post
-reason: String
-text: String
Event
+location: String
+attendees: List<Person>
Announcement
Demonstration
March
Speech
-speakers: List<Person>
Meeting
Person
-name: String
-organizationsAndRoles: HashMap<Organization, Role>
-password: String
+addOrganization(organization: Organization, role: Role): void
+leaveOrg(organization: Organization): void
GUI
-person: Person
+organizationBox: OrganizationBox
+setupPerson(): Person
+isLoginValid(): boolean
+addOrganization(button: Button): void
Login
-credentials: TreeMap<String, Person>
+findLogin(username: String, password: string): Person
+isLoginValid(person: Person): boolean
+serialize(): void
+deserialize(): void
OrganizationBox
-person: Person
+addOrganization(button: Button, vbox: VBox): void
+backButton(button: Button, vbox: VBox): void
Role
+attendEvent(event: Event, person: Person): void
+readAnnouncement(announcement: Announcement): void
Member
Recruiter
+tryToRecruit(target: Person, organization: Organization): boolean void
+recruit(recruitee: Person, organization: Organization): void
«interface»
Organize
+organizeEvent(organization: Organization, reason: String, text: String, location: String): void
+removeEvent(organization: Organization, event: Event): void
Organizer
-events: List<Event>
+addEvent(event: Event): void
+removeEvent(event: Event): void
+addAttendee(event: Event, person: Person): void
+removeAttendee(event: Event, person: Person): void
President
+kickMember(organization: Organization, person: Person): void
+changePosition(organization: Organization, person: Person, role: Role): void

How we Achieved our Goals (Functionality)

ABILITY TO JOIN OR CREATE NEW ORGANIZATIONS

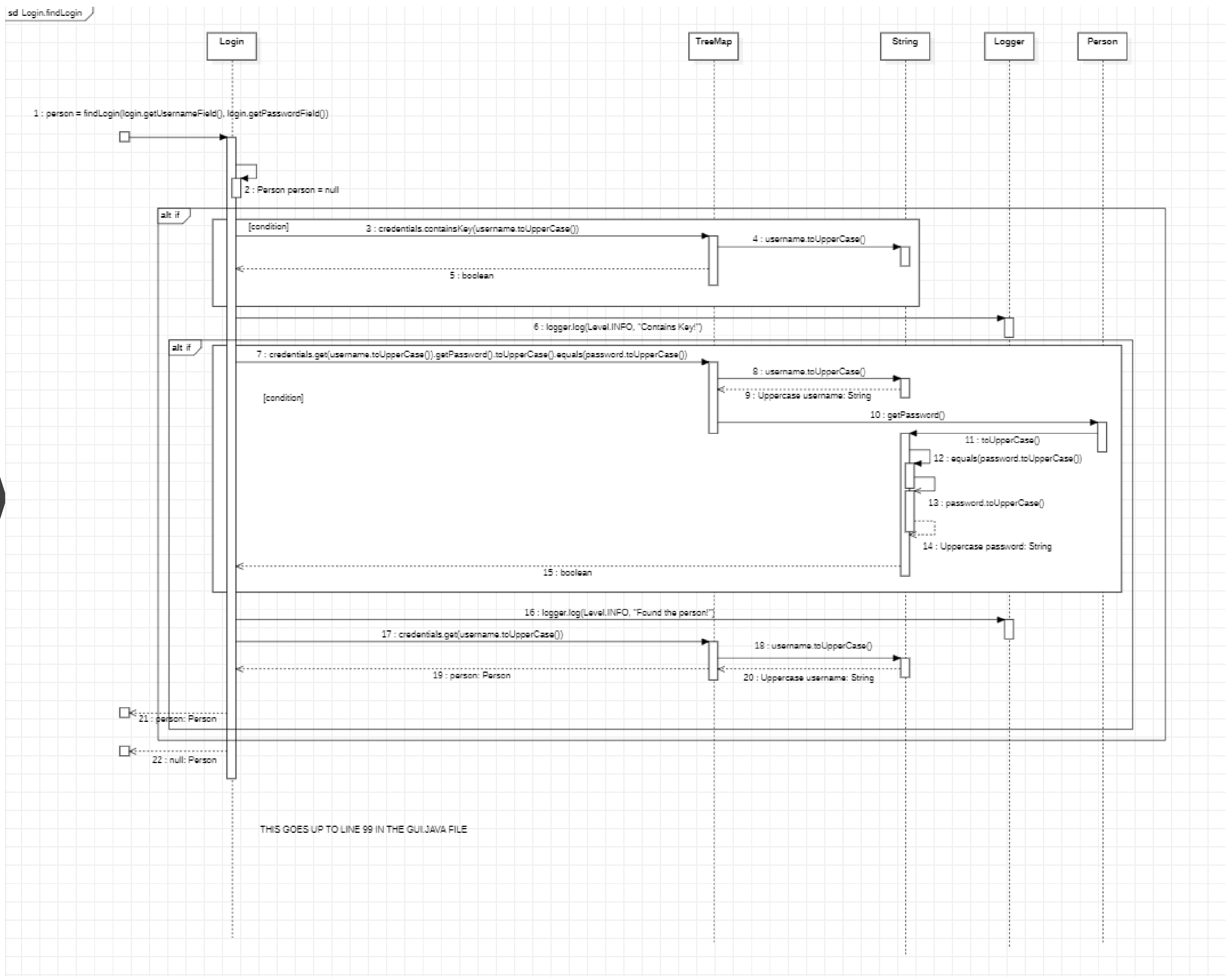
SEARCH VIA TAGS TO FIND SIMILAR ORGANIZATIONS

OPTIONS TO CREATE AND DISTRIBUTE EVENTS AND ANNOUNCEMENTS

ABILITY TO CHANGE ROLES OF INDIVIDUAL MEMBERS TO DELEGATE RESPONSIBILITY

MEMBERS AUTOMATICALLY SEE EVENTS AND ANNOUNCEMENTS UPON LOGGING IN

UML Sequence Diagram


sd Login.findLogin
Login
TreeMap
String
Logger
Person
1 : person = findLogin(login.getUsernameField(), login.getPasswordField())
2 : Person person = null
alt if
[condition]
3 : credentials.containsKey(username.toUpperCase())
4 : username.toUpperCase()
5 : boolean
6 : logger.log(Level.INFO, "Contains Key!")
alt if
7 : credentials.get(username.toUpperCase()).getPassword().toUpperCase().equals(password.toUpperCase())
8 : username.toUpperCase()
9 : Uppercase username: String
[condition]
10 : getPassword()
11 : toUpperCase()
12 : equals(password.toUpperCase())
13 : password.toUpperCase()
14 : Uppercase password: String
15 : boolean
16 : logger.log(Level.INFO, "Found the person!")
17 : credentials.get(username.toUpperCase())
18 : username.toUpperCase()
19 : person: Person
20 : Uppercase username: String
21 : person: Person
22 : null: Person
THIS GOES UP TO LINE 99 IN THE GUI.JAVA FILE

Polymorphic Collections

Each Organization object had a polymorphic collection in the form of a List of Post objects

Post is the generic superclass for any individual Event or Announcement

Each Person has a HashMap of Organizations to Roles

Abstract Classes and Interfaces

Role	an abstract class used for shared functionality of the subclasses like attending events
Interfaces	Organize – enforced methods for managing events
Necessity	This was not particularly necessary, but would be beneficial for scaling up the project
Benefits	Organizing functionality like this also helped ensure SRP compliance

JavaFX GUI

- We utilized a structure similar to Della's House of Bagels to organize our GUI
- Two separate GUIs, GUI and LoginGUI, each with various custom boxes
- Custom boxes

Custom Error Handling

- UnknownFileExtensionException
  - Used to track when incorrect files are input
  - Gives the time stamp and file name
  - Tells user that only .txt and .ser files are accepted

File I/O and Serialization

Organizations have the ability to save a list of all members and their role within that organization

The list of members could also be set using a text file

Organizations could also be serialized as a whole which would save events and organizations as well

To help with the enforcement of our File I/O methods, we utilized the custom FileExtensionNotFound

Logging

Logging was implemented in every class to make debugging substantially easier

Each method would log something whether it succeeded or failed

By implementing it this way, debugging was made easier because it would clarify which method failed and why

Because of this, we could also know if the method never ran at all because no log would appear

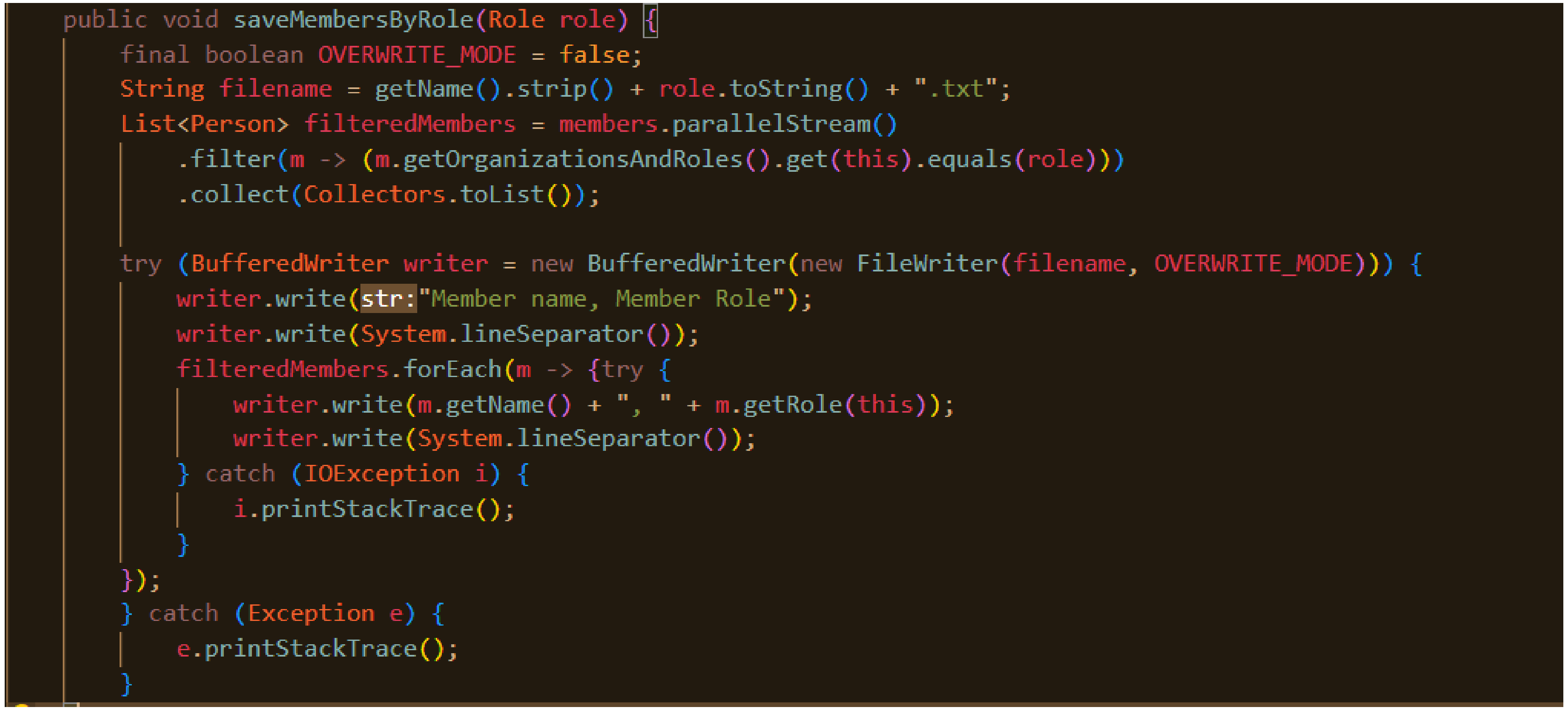
Lambda Expressions

Used both as instance variables and in streams for filtering

We primarily utilized Predicates to filter based on certain criteria

Streams

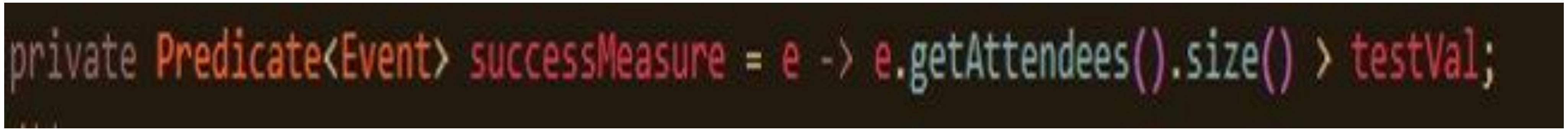
- Streams were used to create methods that allowed for members to be searched for by their role and output to a file



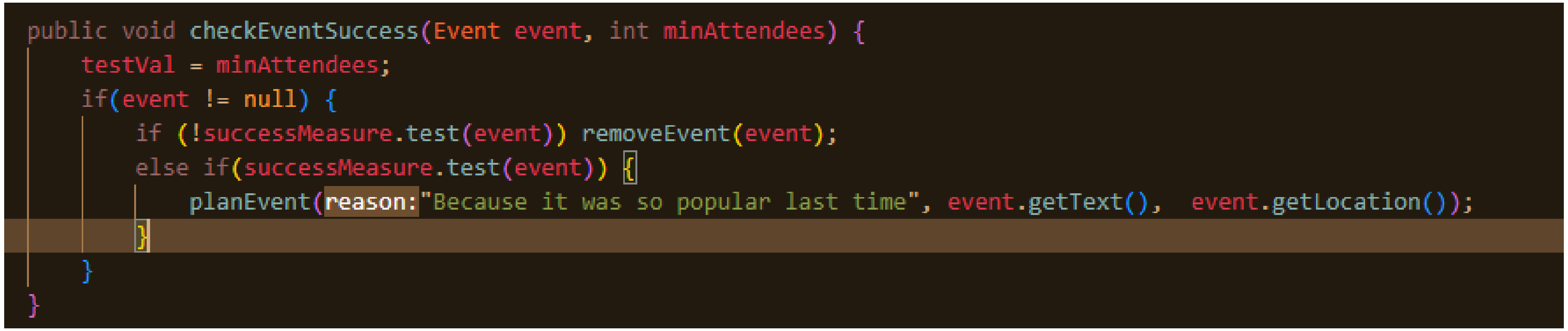
```
public void saveMembersByRole(Role role) {
    final boolean OVERWRITE_MODE = false;
    String filename = getName().strip() + role.toString() + ".txt";
    List<Person> filteredMembers = members.parallelStream()
        .filter(m -> (m.getOrganizationsAndRoles().get(this).equals(role)))
        .collect(Collectors.toList());

    try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename, OVERWRITE_MODE))) {
        writer.write(str:"Member name, Member Role");
        writer.write(System.lineSeparator());
        filteredMembers.forEach(m -> {try {
            writer.write(m.getName() + ", " + m.getRole(this));
            writer.write(System.lineSeparator());
        } catch (IOException i) {
            i.printStackTrace();
        }
    });
    } catch (Exception e) {
        e.printStackTrace();
    }
```

Here used to Measure Event Success



```
private Predicate<Event> successMeasure = e -> e.getAttendees().size() > testVal;
```



```
public void checkEventSuccess(Event event, int minAttendees) {
    testVal = minAttendees;
    if(event != null) {
        if (!successMeasure.test(event)) removeEvent(event);
        else if(successMeasure.test(event)) {
            planEvent(reason:"Because it was so popular last time", event.getText(),  event.getLocation());
        }
    }
}
```

Design Patterns (if any)

Had we learned about design patterns earlier, the MVC pattern would have been our go-to

Since our project has a very distinct front end and back end, MVC would have given us a very nice structure for designing our classes

While we did not struggle much with inheritance, our project likely could have still benefitted from a concrete structure

Threads

- We failed to implement threading due to time crunch, however, here is how we planned to implement it
- We intended to make a side pane in the main GUI that showed similar Organizations that you were a part of
- This would have functioned similarly to Social Media with one thread keeping track of the counts using on the Purpose enum
- Due to issues getting our GUI to work as intended, this had to be scrapped

Challenges we Faced

Beginning the project too late

Communicating effectively

Delegating responsibility

Divide between project in theory and in practice

What we learned

Interesting ways to use files to store and retrieve information

How to structure a GUI in an object-oriented fashion

New and clever ways to use logging to benefit ourselves

Overall project design and planning, although we struggled, we now are aware of difficulties that we previously overlooked

Team Contributions

Ana

GUI

Design Planning

Helper Methods/Classes

Base Classes

Error Handling and Custom Exceptions

Sequence Diagram

Christian

File I/O
Serialization
GUI
Helper Classes
Button handling in the main GUI

Logging

Javadoc

Login class

Jimmy

Logging

Streams

File I/O

Interfaces

Abstract Classes

Lambdas

Role subclasses

Class diagram

Michael

Made main GUI functional

On-click for Organizations to display different controls

Search box in GUI

Add/Remove event buttons

Owen

Theme Creation

Project Direction

Planning

GUI

Helper Classes

Sequence diagram