Table of Contents

Table of figures

Abstract

This report presents the analysis, design and implementation for – “Social Time”, an event planner system developed in object-oriented paradigm. The report is divided into 5 sections, which collectively documents the analysis, design, and implementation of the system. Each section is accompanied by code snippets.

Introduction

“Social Time’ sets itself apart from the general event planner tools in that it is attendee oriented. The primary reason is that the event date is algorithmically set to the best of feasible timeslots in the attendee. This report was written as part of the assignment (June 2017) of L5DC’s module – ‘ADI (Analysis, Design, & Implementation)’. The report documents each of the system development stages namely – analysis, design, and, implementation.

Firstly, use cases for the system were identified which is part of Task 3 (Use Case Diagram). Then, as part of the requirement analysis, Natural Language Analysis (NLA), was conducted to represent domain level class diagram for the system. This is shown in Task 1 (List of Potential Class & Diagram). Then, dynamic model of the system was designed in Task 2 (Activity Diagram), where the core logic of feasible timeslot calculation is presented. The final static architecture is presented in Task 4. This contains the class diagram of the final system that includes persistence layer, core business, and presentation layer with design patterns applied. Lastly, the implementation was done in Java 8 using Netbeans 8.1 with JDK version 1.8.0_121.

Task 1 – List of potential/candidate classes with final class diagram

NLA

NLA abbreviated for Natural Language Analysis is the process of requirement analysis of system on a given written-documentation of scenario or brief. The goal of NLA is to identify business models in the system. And, to obtain a starting static model of the system represented as class diagram. NLA involves steps for filtrations for nouns as potential classes and attributes, adjectives as attributes (not always explicitly defined.), and verbs as potential methods.

Step 1 - Noun identification for potential classes

- Scenario brief was studied thoroughly and all nouns were identified.
- Filtration was applied on nouns to get final list of potential classes.
    - Redundant nouns were removed.
    - Synonymous nouns were removed.
    - Data carrying capacity of noun was checked.
        - If many, then noun was selected for class (Example: User has many data members like name, location etc)
        - If 1, then noun was selected for attribtute. (Example: Data can hold only 1 value i.e. date).
    - Architecture level actions and objects were skipped
        - Example: databases and views
    - Future inference into actions were skipped
        - Generating uncertain timeslots based on available & unavailable timeslots. (This is done algorithmically with correct parameters and need no representation in domain level Class diagram.)

Step 1: Noun identification for potential class:

#	Noun identification for candidate class.	Selected as candidate class?	Justification for selection or rejection as candidate class.
1	meeting	No	Immaterial to the system. Event supersedes metting.
2	friends	No	Social relation is immaterial to the system.
3	family	No	Social relation is immaterial to the system.
4	people	No	Concept found in high level of hyponym heirarchy thus manifesting extreme abstraction.
5	local government initiative	No	Immaterial to the system.
6	time*	No	It is a well defined value and can be an attribute of class - 'Timeslot', to represent hourly blocks.
7	social obligation	No	Immaterial to the system.
8	**social time**	**Yes**	It is the system's proper name. It can act as a façade to wrap rest of the classes in the system. It can contain methods that does not belong to the model classes. However, in the re-design phase of class diagram, it can be replaced by component facades.
9	invitation	No	Invitation need not separate class representation, as it is composed of existing classes's attributes like Event's details. This can be programmatically generated in future as a observer notification.
10	interest	No	The absence and presence of invited attendees in the collection of attendees can be used to infer

11	application	No	The whole architecture makes up the application. It cannot be an individual class.
12	populat slot	No	It is determined by the system in future. It is picked by the system from the maximum voted timeslot and need no representation as separate class.
13	**Vote**	**Yes**	It can contain attributes such as voter (of type - 'RegisteredUser'), the event (of type - 'Event') the user voted to and the timeslot that was voted on.
14	algorithm	No	It is part of brief's context for explaination of implementation details. It appears in context of overriding application's algorithm when user votes on a certain timeslot.
15	profile	No	Profile is made of user attributes and relation to event and timeslot class. This is represented in the view in future without separate class.
16	user	No	It can be replaced by a more specific class - 'RegisteredUser'.
17	**user**	**Yes**	It was analysed that 'RegisteredUser', is the top-level entity in the domain's hyponym heirarchy of actors. It can be an abstract superclass with many attributes such as id, name, email, location, & schedule of 1 week, and collection of created events.

18	attendees	No	All invitee who accepts the invite is taken as attendees. No need for separate representation. This is maintained as attribute of event class.
19	potential attendees	No	All registered users are potential attendees.
20	email*	No	It is a well defined value and is best represented as an attribute of the class - 'RegisteredUser'
21	schedule	No	Schedule represents available, unavailable, and uncertain timeslots. These are attributes of user class. Separate class and association is unncessary.
22	**timeslot**	**Yes**	This is a fundamental class in the system. Weekly recurring schedule of a user consists of 24 timeslots for each day. Each timeslots consists of attributes such as start time, end time and day.
23	available timeslot	No	It is 1 of the values of Timeslot's type attribute.
24	unavailable timeslot	No	It is 1 of the values of Timeslot's type attribute.
25	**candidate timeslot**	**Yes**	It will hold feasible timeslots - available and uncertain, and their vote counts. Fasible timeslots need to be calculated across all invitees and need to be displayed where they can vote.
26	**event**	**Yes**	A fundamental class that will hold various attributes such as id, name, location description, & minimum no of attendees, minimum threshold, invited attendees, and attendees.

27	date*	No	It is a well defined value and is best represented as an attribute of the class - 'Event'
28	system*	No	The whole architectural elements makes up the system. It cannot be an individual class.
29	name*	No	It is a well defined value and is best represented as an attribute of the class - 'Event'
30	location*	No	It is a well defined value and is best represented as an attribute of the class - 'Event' and 'RegisteredUser'
31	description*	No	It is a well defined value and is best represented as an attribute of the class - 'Event'
32	duration*	No	It is a well defined value and is best represented as an attribute of the class - 'Event'
33	**invited attendee (invitee)**	**Yes**	Invited Attendee is a sub class of super class - 'RegisteredUser'. They have specific methods for responding to invitation and voting on timeslots. It will also reside in event class in its attribute of collection type, so that event wise invitees can be tracked.
34	attendees minimum no.*	No	It is a well defined value and is best represented as an attribute of the class - 'Event'
35	minimum threshold*	No	It is a well defined value and is best represented as an attribute of the class - 'Event'
Note: nouns marked with * are attributes			

Adjectives

There were no explicit definitions of Adjectives in the scenario, that maps to attribute of class. Hence, Attributes are inferred from the nouns itself.

Fitered list of candidate class	
#	Selected class
1	User
2	Invitee
3	Event
4	Timeslot
5	CandidateTimeslot
6	SocialTime

Step 2 – Verb identification for potential methods/operations

- Filtration was applied on verbs to get final list of potential methods.
    - Redundant verbs were removed.
    - Synonymous verbs were removed.
    - Architecture level actions and objects were skipped
        - Example: saving to databases and creating views
    - Future inference into actions were skipped
        - Generating uncertain timeslots based on available & unavailable timeslots. (This is done algorithmically with correct parameters and need no representation in domain level Class diagram.)

Step 2 - Verb identification for potential operations			
#	**Verbs as potential behavior**	**Can it be an actual method?**	**Justification**
1	arrange meetings	No	It comes in context of the people's problem stated in the scenario as in - 'difficulty to arrange meeting'. It is not part of specific behavior of any architectural component.
2	organise time	No	It comes in context of the people's problem stated in the scenario as in - 'difficulty to organize time'. It is not part of specific behavior of any architectural component.
3	track social obligation	No	It is conceptual function of the whole system. It is not a specific behavior of any architectural component.
4	**create event**	**Yes**	It is a fundamental behavior of the registered user of this system.
5	suggest date-time	No	It is generalization of 3 operations for suggesting suitable timeslots to user. The specific operations are - 'calculate feasible timeslot', 'generate canidate timeslot list'
6	enter user information	No	This is an action performed by registered user. User details need to be saved by the system. This is achieved through input via UI elements and data storage via several mutator methods (setters) in the 'User' class.
7	**show invitation**	**Yes**	A method of the main class - 'SocialTime', which displays invitation details to the invitee.
8	express interest	No	Synonymous to the operation, 'accept invitation' of invited users.
9	dismiss invitation	No	Synonymous to the operation 'reject invitation'.
10	provide date-time list	No	Synonymous to the operation 'present timeslot list'.
11	indicate attendance desire	No	A general operation of invited users & can be replaced by more specific operations - 'accept invitation' & 'reject invitation'

12	compare time-slots	No	It is a general operation of timeslot handler class and can be replacable by a more specific operation - 'calculate feasible timeslots'
13	provide appropriate time-slot list	No	Synonymous to the operation 'present timeslot list'.
14	**vote on timeslot**	**Yes**	Once the invitee accepts the invite, user should register their vote to the timeslot. This is an important operation of invitee that will determine the final date of event.
15	over-ride algorithm	No	This comes in context of allowing user to update their uncertain timeslot if it is the timeslot that was voted for. It is part of the operation - 'vote on timeslot'.
16	**pick popular timeslot**	**Yes**	This method belongs to Timeslot handler must count the votes and pick the timeslot with maximum vote. This timeslot would then be the event's date.
17	**set event date-time**	**Yes**	It is the setter method for Event class. After the populat timeslot is picked, this method is used to set event's date.
18	**register user**	**Yes**	For domain level class diagram, it will be a method of class 'Social Time'. Later, in the redesign, it will most likely be represented in 'FacadeUser' class associated to 'UserDAO' (Data Access Object for for user details' persistence.
19	edit profile	No	This operation need not representation as mutator methods/setters in 'RegisteredUser' class suffices its roles
20	**add event (create event)**	**Yes**	For domain level class diagram, it will be a method of class 'SocialTime'. Later, in the redesign it will most likely be represented in 'FacadeEvent' class associated to the factory class for event instantiation.

21	edit event	No	This operation need not representation as mutator methods/setters in 'Event' class suffices its roles
22	**view event-list**	**Yes**	UserController's operation. For now it will reside in Social Time class. This is one of the response actions of invitee.
23	**accept invitation**	**Yes**	UserController's operation. For now it will reside in Social Time class. This is one of the response actions of invitee.
24	**reject invitation**	**Yes**	UserController's operation. For now it will reside in Social Time class. This is one of the response actions of invitee.
25	calculate feasible timeslots	No	Replaced by a more specific method - generate candidate timeslot list'
26	**generate candidate timeslot list**	**Yes**	This will be a method of timeslot controller This is the core logic of the system where across all invitees, the feasible timeslot is extracted. In the domain level class diagram, it is represented in Social Time.
27	**present timeslot list**	**Yes**	This will be a method of timeslot controller. In the domain level class diagram, it is represented in Social Time.
28	finalise event details	No	Setters and Getters will suffice this rule as finalization of event details means to set the event date after the voting is conducted.

Fitered list of candidate methods

#	Selected methods
1	create event
2	show invitation
3	vote on timeslot
4	pick popular timeslot
5	set event date-time
6	register user
7	add event (create event)
8	view event-list
9	accept invitation
10	reject invitation
11	generate candidate timeslot list
12	present timeslot list

Step 3 – Grouping methods and attributes to domain classes

Allocating finalized attributes and methods (from step 1 and 2) respectice finalized classes (from step 1)				
#	**Class**	**Attribute**	**Method**	
1	User	• id • name • location • email • location • createdEvents • availableTimeSlots • unavailableTimeSlots	*None*	
2	Invitee	• (Inherits from 'RegisteredUser' class) • Invitations	• viewInvitedEventList() • acceptInvitation() • rejectInvitation() • voteOnTimeSlot()	
3	Event	• id • name • location • description • date • duration • minNoOfAttendees • minThresholdPercent • invitedAttendees • attendees		
6	Timeslot	• day • hour	*None*	
7	CandidateTimeslot	• (Inherits from 'Timeslot' class) • voteCount	*None*	
8	SocialTime	*None*	• createEvent() • registerUser() • generateCandidateTimeslotList() • presentTimeslotList() • pickPopularTimeslot()	

Step 4 – Drafting class diagram

Finally, from the NLA above, domain level class diagram is drafted. This level of diagram only shows the relationship between the business models and identifies core logic only. Here, persistence, and presentation layer is not shown.

Reference for interpreting presented class diagrams

- Diagram is based on **UML version 2.X**
- Modelling tool used was **Visual Paradigm Community Edition 14.0 (Build 20170302)**

Notation	Remark
Generalization	Depicts inheritance from super/parent class to sub/child class.
Realization	Depicts interface implementation.
	Depicts unidirectional association between classes.
Association	Depicts bidirectional association between classes.
Aggregation	Depicts 1 class aggregates another without binding its existence in own life.
Composition	Depicts 1 class composes another binding its existence in its own life.
Dependency	Depicts 1 class depends on another in such way that the change propagates to dependent.

Figure 1:Class Diagram Legend

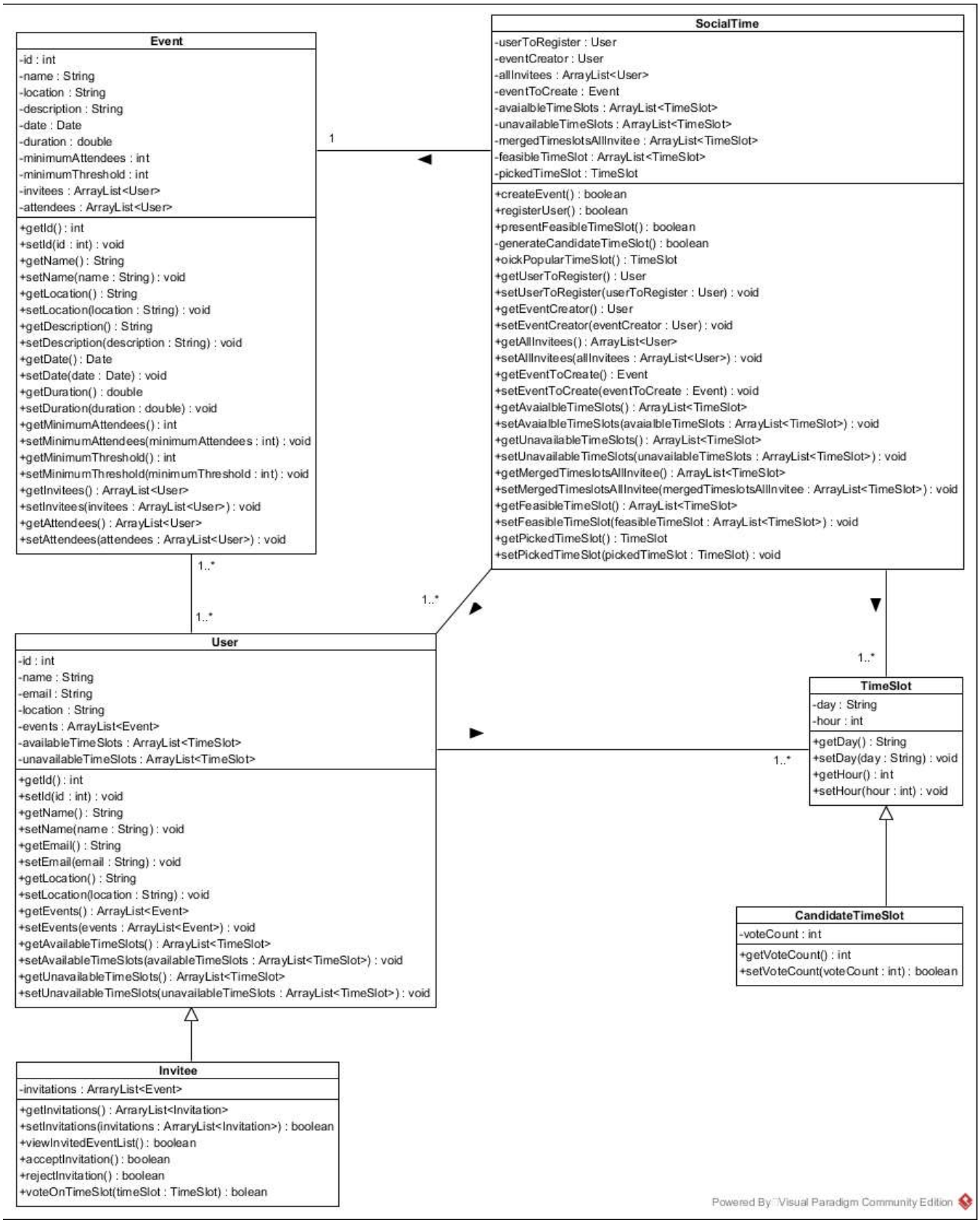

Figure 2: Class Diagram (Domain Level)

Task 2 – Activity Diagram of feasible timeslot

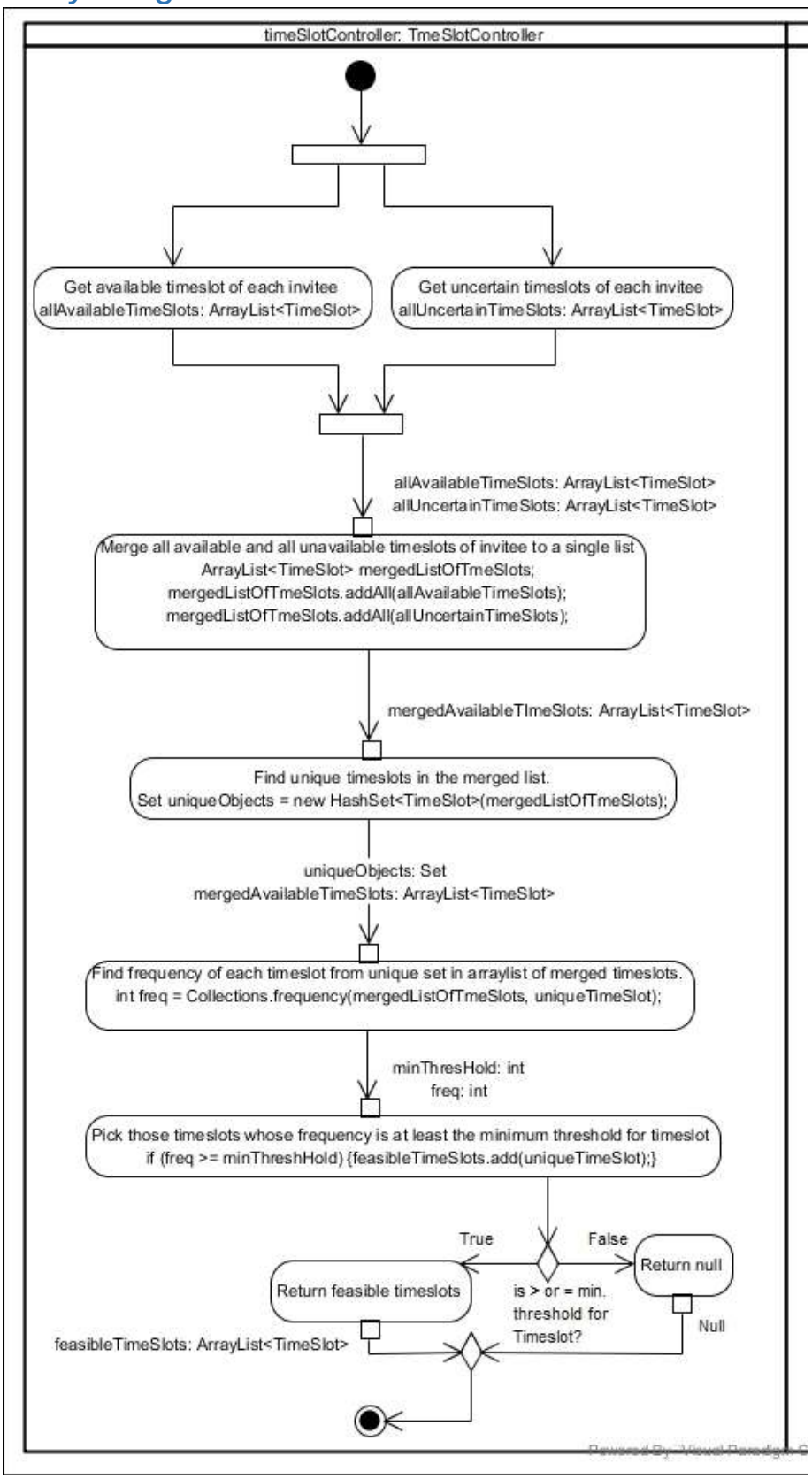

Figure 3: Activity Diagram for feasible timeslot generation

Task 3 – Use Case

Use Case

Brugge and Dutoit, in their book “Object-oriented software engineering: Using UML, Patterns, and Java” (Bruggee et al, 2010), introduces use case as a tool for both requirement analysis and behavioural modelling. Use case’s goal is to depict relationship between actors and the system via well-defined achievable functions known as use case. Actors are role abstractions and not human users.

Following diagram depicts the use case for “Social Time”. All attendees are invitees, and both attendees and invitees are users or registered users. Each relate to the system and associate with particular use cases as shown below.

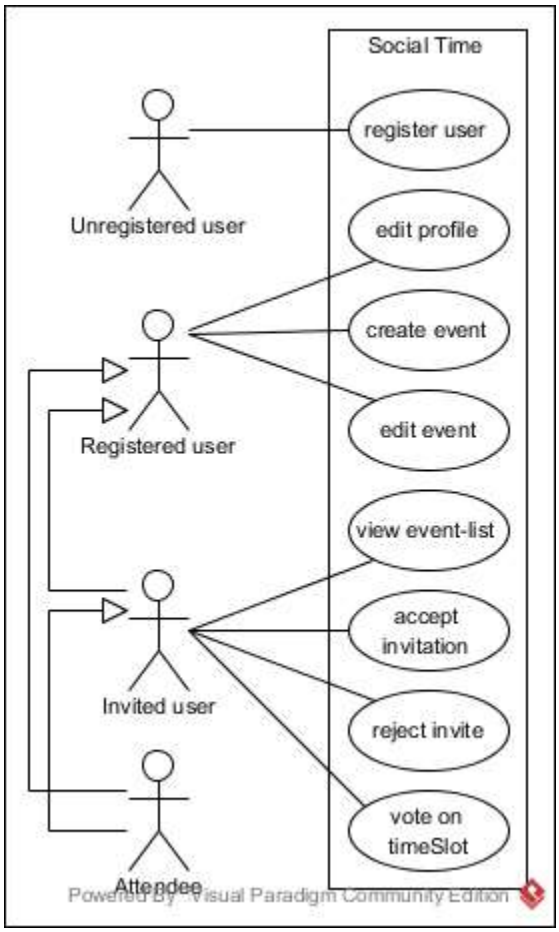

Figure 4: Use Case - Social Time

Task 4 – Code Architecture

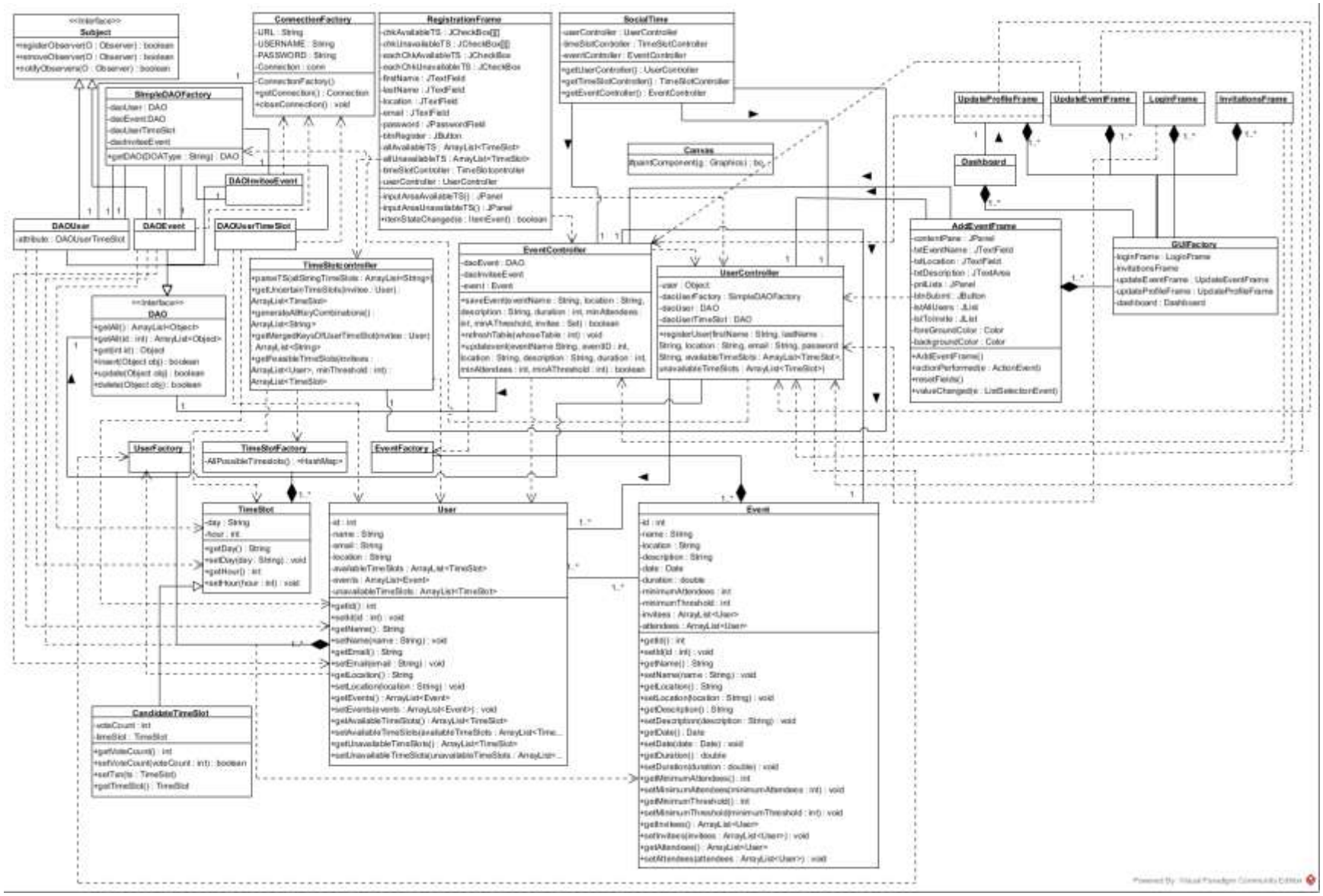

Figure 5: Class Diagram (Architecture Level)

Task 5 – System implementation: Java source code

Package – controller

EventController.java

```java
package controller;

import dao.DAO;
import dao.SimpleDAOFactory;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Set;
import model.Event;
import model.User;
import model.factory.EventFactory;

/**
 *
 * @author Biju Ale
 */
public class EventController {

    private DAO daoEvent, daoInviteeEvent;
    private Event event;

    public boolean saveEvent(String eventName, String location, String description, int duration, int minAttendees, int minAThreshold, Set invitee) {

        //Parse set to arraylist and remove null
        ArrayList<User> invitees = new ArrayList();
        invitees.addAll(invitee);
        invitees.removeAll(Collections.singleton(null));

        //Persist invitee to DB
        for (User invitee1 : invitees) {
            System.out.println(invitee1.getId());
        }

        //Get event object
        event = EventFactory.getModel(UserController.getLoggedInUser(), eventName, location, description, duration, minAttendees, minAThreshold, invitees);
        //Get DAOEvent
        daoEvent = SimpleDAOFactory.getDAO("DAOEvent");

        //Persist event to DB using DAOEvent
        daoEvent.insert(event);

        event.setInvitee(invitees);
```

```
47
48          daoInviteeEvent = SimpleDAOFactory.getDAO("DAOInviteeEvent");
49          daoInviteeEvent.insert(event);
50          refreshTable(event.getCreator().getId());
51          return true;
52      }
53
55      public void refreshTable(int whoseTable) {
55          daoEvent = SimpleDAOFactory.getDAO("DAOEvent");
56          daoEvent.getAll(whoseTable);
57      }
58
59      public boolean updateEvent(String eventName, int eventID, String location, String ↩
    description, int duration, int minAttendees, int minAThreshold) {
60          //Get event object
61          event = EventFactory.getModel(eventID, UserController.getLoggedInUser(), ↩
    eventName, location, description, duration, minAttendees, minAThreshold);
62          //Get DAOEvent
63          daoEvent = SimpleDAOFactory.getDAO("DAOEvent");
64          //Persist event to DB using DAOEvent
65          daoEvent.update(event);
66          //Persist event in DB
67          refreshTable(event.getCreator().getId());
68          return true;
69      }
70
71  }
72
```

SocialTime.java

```java
package controller;

/**
 *
 * @author Biju Ale
 */
public class SocialTime {

    private static UserController userController;
    private static TimeSlotController timeSlotController;
    private static UserTimeSlotController userTimeSlotController;
    private static EventController eventController;

    //Getters using singleton pattern to instantiate and return controller objects
    public static UserController getUserController() {
        if (userController == null) {
            userController = new UserController();
        }
        return userController;
    }

    public static TimeSlotController getTimeSlotController() {
        if (timeSlotController == null) {
            timeSlotController = new TimeSlotController();
        }
        return timeSlotController;
    }

    public static UserTimeSlotController getUserTimeSlotController() {
        if (userTimeSlotController == null) {
            userTimeSlotController = new UserTimeSlotController();
        }
        return userTimeSlotController;
    }

    public static EventController getEventController() {
        if (eventController == null) {
            eventController = new EventController();
        }
        return eventController;
    }

}
```

TimeSlotController.java

```java
package controller;

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Set;
import model.TimeSlot;
import model.User;
import model.factory.TimeSlotFactory;

/**
 *
 * @author Biju Ale
 */
public class TimeSlotController {

    //Returns list of timeslot objects parsed from string
    public ArrayList<TimeSlot> parseTS(ArrayList<String> allStringTimeSlots) {
        TimeSlot timeSlot;
        ArrayList<TimeSlot> allParsedTimeSlots = new ArrayList();

        for (String eachStringTimeSlot : allStringTimeSlots) {
            String tokens[] = eachStringTimeSlot.split(";");
            int day = Integer.parseInt(tokens[0]);
            int hour = Integer.parseInt(tokens[1]);
            timeSlot = TimeSlotFactory.getTimeSlot(day, hour);
            allParsedTimeSlots.add(timeSlot);
        }
        return allParsedTimeSlots;
    }

    public ArrayList<TimeSlot> getUncertainTimeSlots(User invitee) {
        ArrayList<TimeSlot> uncertainTimeSlots = new ArrayList();
        TimeSlot uncertainTimeslot;

        ArrayList<String> allKeys = generateAllKeyCombinations();
        ArrayList<String> mergedKeys = getMergerdKeysOfUserTimeSlot(invitee);//Available+Unavailble
        ArrayList<String> uncertainTimeSlotKeys = new ArrayList();
        allKeys.removeAll(mergedKeys);

```

```java
        uncertainTimeSlotKeys.addAll(allKeys);//allKeys now contain only uncertain timeslot keys.

        HashMap<String, Object> timeSlotParameters;

        //Get uncertain timeslot objects from factory based on all uncertain timeslot keys
        for (String uncertainTimeSlotKey : uncertainTimeSlotKeys) {
            String tokens[] = uncertainTimeSlotKey.split(";");
            int day = Integer.parseInt(tokens[0]);
            int hour = Integer.parseInt(tokens[1]);
            uncertainTimeslot = TimeSlotFactory.getTimeSlot(day, hour);
            uncertainTimeSlots.add(uncertainTimeslot);
        }
        return uncertainTimeSlots;
    }

    //Generate all possible day-hour combination for timeslots (available+unavailable+uncertain)
    public ArrayList<String> generateAllKeyCombinations() {
        ArrayList<String> allKeys = new ArrayList();
        for (int dayNo = 1; dayNo < 8; dayNo++) {
            for (int hour = 0; hour < 24; hour++) {
                String key = Integer.toString(dayNo) + ";" + Integer.toString(hour);
                allKeys.add(key);
            }
        }
        return allKeys;
    }

//    Get merged keys (available+unavailable timeslot) of user.
    public ArrayList<String> getMergerdKeysOfUserTimeSlot(User invitee) {
        ArrayList<String> mergedKeys = new ArrayList();

        ArrayList<TimeSlot> mergedList = new ArrayList();
        mergedList.addAll(invitee.getAvailableTimeSlots());
        mergedList.addAll(invitee.getUnavailableTimeSlots());

        for (TimeSlot timeSlot : mergedList) {
            String key = Integer.toString(timeSlot.getDay()) + ";" + Integer.toString(timeSlot.getHour());
            mergedKeys.add(key);
```

```java
        }
        return mergedKeys;
    }

    public ArrayList<TimeSlot> getFeasibleTimeSlots(ArrayList<User> invitees, int minThreshHold) {
        ArrayList<TimeSlot> feasibleTimeSlots = new ArrayList();
        ArrayList<TimeSlot> mergedListOfTmeSlots = new ArrayList();//Available & Unncertain timeslots of all invitee merged in 1 list

        ArrayList<TimeSlot> availableTimeSlot, unCertainTimeSlot;

        Set<TimeSlot> uniqueObjects;

        //Add all timeslots of each invitee into a single list
        for (User invitee : invitees) {
            mergedListOfTmeSlots.addAll(invitee.getAvailableTimeSlots());
            mergedListOfTmeSlots.addAll(getUncertainTimeSlots(invitee));
        }

        //Find the unique set from combined arraylist
        uniqueObjects = new HashSet<TimeSlot>(mergedListOfTmeSlots);

        //Check feasibility of timeslots with min threshold.
        for (TimeSlot uniqueTimeSlot : uniqueObjects) {
            int freq = Collections.frequency(mergedListOfTmeSlots, uniqueTimeSlot);
            if (freq >= minThreshHold) {
                feasibleTimeSlots.add(uniqueTimeSlot);
            }
        }
        return feasibleTimeSlots;
    }

}
```

UserController.java

```java
package controller;

import dao.DAO;
import dao.DAOUser;
import java.util.ArrayList;
import model.TimeSlot;
import dao.SimpleDAOFactory;
import model.User;
import model.factory.UserFactory;

/**
 *
 * @author Biju Ale
 */
public class UserController {

    public static User getLoggedInUser() {
        return loggedInUser;
    }

//    private SimpleModelFactory userFactory;
    private Object user;
//    private SimpleDAOFactory daoUserFactory;
    private DAO daoUser, daoUserTimeSlot;
    private static User loggedInUser;

    public boolean registerUser(String firstName, String lastName, String email, String location, String password, ArrayList<TimeSlot> allAvailableTS, ArrayList<TimeSlot> allUnavailableTS) {
        //Get user object
        user = UserFactory.getModel(firstName, lastName, email, location, password, allAvailableTS, allUnavailableTS);
        //Get DAOUser
        daoUser = SimpleDAOFactory.getDAO("DAOUser");
        //Persist user to DB using DAOUser
        daoUser.insert(user);
        //Get DAOUserTimeSlot
        daoUserTimeSlot = SimpleDAOFactory.getDAO("DAOUserTimeSlot");
        //Persist user and timeslot details to DB using DAOUserTimeSlot
        daoUserTimeSlot.insert(user);
        return true;
    }
```

```
45
46      public boolean authenticateUser(String email, String password) {
47          loggedInUser = (User) ((DAOUser) SimpleDAOFactory.getDAO("DAOUser")).get(email,↩
    password);
            if (loggedInUser.getId() == 0) {
49              return false;
50          } else {
51              return true;
52          }
53      }
54
55      public void updateProfile(User loggedInUser) {
56          //Get DAOUser
57          daoUser = SimpleDAOFactory.getDAO("DAOUser");
58          //update user to DB using DAOUser
59          daoUser.update(loggedInUser);
60      }
61  }
62
```

Package – dao

DAO.java (Interface)

```java
package dao;

import java.util.ArrayList;

/**
 *
 * @author Biju Ale
 */
public interface DAO {
    ArrayList<Object> getAll();
    ArrayList<Object> getAll(int id);
    Object get(int id);

    boolean insert(Object obj);
    boolean update(Object obj);
    boolean delete(Object obj);
}
```

DAOEvent.java

```java
package dao;

import dbutil.ConnectionFactory;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import javax.swing.JOptionPane;
import model.Event;
import model.User;
import subject_observer.Observer;
import subject_observer.Subject;
import view.Dashboard;

/**
 *
 * @author Biju Ale
 */
public class DAOEvent implements DAO, Subject {

    Connection conn;
    //private Dashboard dashboard = (Dashboard) GUIFactory.getInstanceOf("dashboard'
    public static Dashboard dashboard;
    private ArrayList events = new ArrayList();

    public DAOEvent() {
        conn = ConnectionFactory.getConnection();
    }

    @Override
    public ArrayList<Object> getAll() {
        String sql = "SELECT * FROM tbl_event";

        events.clear(); //Clear history
        try {
            PreparedStatement stmt = conn.prepareStatement(sql);
            ResultSet rs = stmt.executeQuery();
            while (rs.next()) {
                Event event = new Event();
```

```
47                  event.setId(rs.getInt("id"));
48                  event.setName(rs.getString("name"));
49                  event.setLocation(rs.getString("location"));
50                  event.setDescription(rs.getString("description"));
51                  event.setDuration(rs.getInt("duration"));
52                  event.setDate(rs.getDate("date"));
53                  event.setMinNoOfAttendees(rs.getInt("minNoOfAttendees"));
54                  event.setMinThresholdPercent(rs.getInt("minThresholdPercent"));
55                  events.add(event);
56              }
57              // Build table model in dashboard
58  //            dashboard.updateTable(events);
59  //            notifyObserver(dashboard);
60              stmt.close();
61              rs.close();
62
63          } catch (SQLException ex) {
64              JOptionPane.showMessageDialog(null, ex.getStackTrace(), ex.getMessage(), ↩
    JOptionPane.ERROR_MESSAGE);
65          }
66          return events;
67      }
68
69      @Override
        public Object get(int id) {
71          String sql = "SELECT * FROM tbl_event WHERE id = ?";
72          Event event = new Event();
73          try {
                PreparedStatement stmt = conn.prepareStatement(sql);
75              stmt.setInt(id, id);
76              ResultSet rs = stmt.executeQuery(sql);
77              if (rs.next()) {
78                  event.setName(rs.getString("name"));
79                  event.setLocation(rs.getString("location"));
80                  event.setDescription(rs.getString("description"));
81                  event.setDuration(rs.getInt("duration"));
82                  event.setDate(rs.getDate("date"));
83                  event.setMinNoOfAttendees(rs.getInt("minNoOfAttendees"));
84                  event.setMinThresholdPercent(rs.getInt("minThrestholdPercent"));
85              }
86              stmt.close();
87              rs.close();
88
```

```
89          } catch (SQLException ex) {
90              JOptionPane.showMessageDialog(null, ex.getStackTrace(), ex.getMessage(), ↩
    JOptionPane.ERROR_MESSAGE);
91          }
92          return event;
93      }
94
95      @Override
96      public boolean insert(Object obj) {
97          String toEventTable = "INSERT INTO tbl_event(creator, name, location, ↩
    description, duration, minNoOfAttendees, minThresholdPercent) VALUES(?, ?,?,?,?,?,?)";↩

98          Event event = (Event) obj;
99          int creatorID = ((User) event.getCreator()).getId();
100         try {
                PreparedStatement stmt = conn.prepareStatement(toEventTable, Statement.↩
    RETURN_GENERATED_KEYS);
102             stmt.setInt(1, (creatorID));
103             stmt.setString(2, event.getName());
104             stmt.setString(3, event.getLocation());
105             stmt.setString(4, event.getDescription());
106             stmt.setDouble(5, event.getDuration());
107             stmt.setInt(6, event.getMinNoOfAttendees());
108             stmt.setInt(7, event.getMinThresholdPercent());
109
110             stmt.executeUpdate();
111             ResultSet rs = stmt.getGeneratedKeys();
112             rs.next();
113             int eventID = rs.getInt(1);//Return auto-generataed key to set ID of object
114             event.setId(eventID);
115             stmt.close();
116             return true;
117         } catch (SQLException ex) {
118             JOptionPane.showMessageDialog(null, ex.getStackTrace(), ex.getMessage(), ↩
    JOptionPane.ERROR_MESSAGE);
119         }
120         return false;
121     }
122
123     @Override
124     public boolean update(Object obj) {
125         String sql = "UPDATE tbl_event SET name = ?, location = ?, description = ?, ↩
```

```
duration = ?, minNoOfAttendees = ?, minThresholdPercent = ? WHERE id = ?";
        Event event = (Event) obj;
        try {

            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setString(1, event.getName());
            stmt.setString(2, event.getLocation());
            stmt.setString(3, event.getDescription());
            stmt.setInt(4, event.getDuration());
            stmt.setInt(5, event.getMinNoOfAttendees());
            stmt.setInt(6, event.getMinThresholdPercent());
            stmt.setInt(7, event.getId());
            int rowsUpdated = stmt.executeUpdate();
            stmt.close();

            if (rowsUpdated > 0) {
//                  notifyObserver(dashboard);
                return true;

            }
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getStackTrace(), ex.getMessage(), ↩
JOptionPane.ERROR_MESSAGE);
        }

        return false;
    }

    @Override
    public boolean delete(Object obj) {
        String sql = "DELETE FROM tbl_event WHERE id = ?";
        Event event = (Event) obj;
        try {
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setInt(1, event.getId());

            int rowsDeleted = stmt.executeUpdate();
            stmt.close();

            if (rowsDeleted > 0) {
                return true;
            }
```

```java
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getStackTrace(), ex.getMessage(), JOptionPane.ERROR_MESSAGE);
        }
        return false;
    }

    @Override
    public ArrayList<Object> getAll(int id) {
        String sql = "SELECT * FROM tbl_event WHERE creator = ?";
        events.clear(); //Clear history
        try {
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setInt(1, id);
            ResultSet rs = stmt.executeQuery();
            while (rs.next()) {
                Event event = new Event();
                event.setId(rs.getInt("id"));
                event.setName(rs.getString("name"));
                event.setLocation(rs.getString("location"));
                event.setDescription(rs.getString("description"));
                event.setDuration(rs.getInt("duration"));
                event.setDate(rs.getDate("date"));
                event.setMinNoOfAttendees(rs.getInt("minNoOfAttendees"));
                event.setMinThresholdPercent(rs.getInt("minThresholdPercent"));
                events.add(event);
            }
            // Build table model in dashboard
            dashboard.updateTable(events);
//              notifyObserver(dashboard);
            stmt.close();
            rs.close();

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getStackTrace(), ex.getMessage(), JOptionPane.ERROR_MESSAGE);
        }
        return events;
    }

```

```java
    @Override
    public boolean registerObserver(Observer O) {
        this.dashboard = (Dashboard) O;
        return true;
    }

    @Override
    public boolean removeObserver(Observer O) {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }

    @Override
    public boolean notifyObserver(Observer O) {
        try {
            dashboard.updateTable(events);
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getStackTrace(), ex.getMessage(), JOptionPane.ERROR_MESSAGE);
        }
        return true;
    }

}
```

DAOInviteeEvent.java

```java
6   package dao;
7
8   import dbutil.ConnectionFactory;
9   import java.sql.Connection;
10  import java.sql.PreparedStatement;
11  import java.sql.ResultSet;
12  import java.sql.SQLException;
13  import java.util.ArrayList;
14  import javax.swing.JOptionPane;
15  import model.Event;
16  import model.User;
17  import view.InvitationsFrame;
18
19  /**
20   *
21   * @author Biju Ale
22   */
23  public class DAOInviteeEvent implements DAO {
24
25      static InvitationsFrame inf;    private ArrayList events = new ArrayList();
26
27      Connection conn;
28
29      public DAOInviteeEvent() {
30          this.conn = ConnectionFactory.getConnection();
31      }
32
33      @Override
34      public ArrayList<Object> getAll(int id) {
35          String sql = "SELECT e. FROM tbl_event WHERE creator = ?";
36          events.clear(); //Clear history
37          try {
38              PreparedStatement stmt = conn.prepareStatement(sql);
39              stmt.setInt(1, id);
40              ResultSet rs = stmt.executeQuery();
41              while (rs.next()) {
42                  Event event = new Event();
43                  event.setId(rs.getInt("id"));
44                  event.setName(rs.getString("name"));
45                  event.setLocation(rs.getString("location"));
46                  event.setDescription(rs.getString("description"));
```

```
47                  event.setDuration(rs.getInt("duration"));
48                  event.setDate(rs.getDate("date"));
49                  event.setMinNoOfAttendees(rs.getInt("minNoOfAttendees"));
50                  event.setMinThresholdPercent(rs.getInt("minThresholdPercent"));
51                  events.add(event);
52              }
53              // Build table model in dashboard
54              inf.updateTable(events);
55  //              notifyObserver(dashboard);
56              stmt.close();
57              rs.close();
58
59          } catch (SQLException ex) {
60              JOptionPane.showMessageDialog(null, ex.getStackTrace(), ex.getMessage(), ↩
    JOptionPane.ERROR_MESSAGE);
61          }
62          return events;
63      }
64
65      @Override
66      public ArrayList<Object> getAll() {
67          throw new UnsupportedOperationException("Not supported yet."); //To change ↩
    body of generated methods, choose Tools | Templates.
68      }
69
70      @Override
71      public Object get(int id) {
72          throw new UnsupportedOperationException("Not supported yet."); //To change ↩
    body of generated methods, choose Tools | Templates.
73      }
74
75      @Override
76      public boolean insert(Object obj) {
77          Event event = (Event) obj;
78          ArrayList<User> invitees = event.getInvitee();
79          String toEventInviteeTable = "INSERT INTO tbl_event_invitee(event_id, user_id)↩
     VALUES(?,?)";
80          try {
81              PreparedStatement stmt = conn.prepareStatement(toEventInviteeTable);
82              //Persisting all invitee with events
83              for (User eachInvitee : invitees) {
84                  stmt.setInt(1, event.getId());
85                  stmt.setInt(2, eachInvitee.getId());
```

```
86                  stmt.executeUpdate();
87              }
88              stmt.close();
89              return true;
90          } catch (SQLException ex) {
91              JOptionPane.showMessageDialog(null, ex.getStackTrace(), ex.getMessage(), ↩
    JOptionPane.ERROR_MESSAGE);
92          }
93          return false;
94      }
95
96      @Override
        public boolean update(Object obj) {
98          throw new UnsupportedOperationException("Not supported yet."); //To change ↩
    body of generated methods, choose Tools | Templates.
99      }
100
101     @Override
        public boolean delete(Object obj) {
103         throw new UnsupportedOperationException("Not supported yet."); //To change ↩
    body of generated methods, choose Tools | Templates.
104     }
105
106 }
107
```

DAOUser.java

```java
6   package dao;
7
8   import dbutil.ConnectionFactory;
9   import java.sql.Connection;
10  import java.sql.PreparedStatement;
11  import java.sql.ResultSet;
12  import java.sql.SQLException;
13  import java.sql.Statement;
14  import java.util.ArrayList;
15  import javax.swing.JOptionPane;
16  import model.User;
17
18  /**
19   *
20   * @author Biju Ale
21   */
22  public class DAOUser implements DAO {
23
24      Connection conn;
25      DAOUserTimeSlot daoUserTimeSlot;
26
27      public DAOUser() {
28          conn = ConnectionFactory.getConnection();
29      }
30
31      @Override
32      public ArrayList<Object> getAll() {
33          String sql = "SELECT * FROM tbl_user";
34          ArrayList<Object> users = new ArrayList();
35          try {
            Statement stmt = conn.createStatement();
37              ResultSet rs = stmt.executeQuery(sql);
38              while (rs.next()) {
39                  User user = new User();
40                  user.setId(rs.getInt("id"));
41                  user.setFirstName(rs.getString("firstName"));
42                  user.setLastName(rs.getString("lastName"));
43                  user.setEmail(rs.getString("email"));
44                  user.setLocation(rs.getString("location"));
45                  user.setPassword(rs.getString("password"));
46                  users.add(user);
47              }
48              stmt.close();
```

```
49                  rs.close();
50
51              } catch (SQLException ex) {
52                  JOptionPane.showMessageDialog(null, ex.getStackTrace(), ex.getMessage(), ↩
    JOptionPane.ERROR_MESSAGE);
53              }
54              return users;
55
56          }
57
58          @Override
59          public Object get(int id) {
60              String sql = "SELECT * FROM tbl_user WHERE id = ?";
61              User user = new User();
62              try {
                    PreparedStatement stmt = conn.prepareStatement(sql);
64                  stmt.setInt(id, id);
65                  ResultSet rs = stmt.executeQuery(sql);
66                  if (rs.next()) {
67                      user.setFirstName(rs.getString("firstName"));
68                      user.setLastName(rs.getString("lastName"));
69                      user.setEmail(rs.getString("email"));
70                      user.setLocation(rs.getString("location"));
71                      user.setPassword(rs.getString("password"));
72                  }
73                  stmt.close();
74                  rs.close();
75
76              } catch (SQLException ex) {
77                  JOptionPane.showMessageDialog(null, ex.getStackTrace(), ex.getMessage(), ↩
    JOptionPane.ERROR_MESSAGE);
78              }
79              return user;
80          }
81
82          //Overload for user authentication
83          public Object get(String email, String password) {
84              String sql = "SELECT * FROM tbl_user WHERE email = ? AND password = ?";
85              User user = new User();
86              try {
                    PreparedStatement stmt = conn.prepareStatement(sql);
88                  stmt.setString(1, email);
89                  stmt.setString(2, password);
```

```
90                ResultSet rs = stmt.executeQuery();
91                while (rs.next()) {
92                    user.setId(rs.getInt("id"));
93                    user.setFirstName(rs.getString("firstName"));
94                    user.setLastName(rs.getString("lastName"));
95                    user.setEmail(rs.getString("email"));
96                    user.setLocation(rs.getString("location"));
97                    user.setPassword(rs.getString("password"));
98                }
99                stmt.close();
100               rs.close();
101
102           } catch (SQLException ex) {
103               JOptionPane.showMessageDialog(null, ex.getStackTrace(), ex.getMessage(), ↩
    JOptionPane.ERROR_MESSAGE);
104           }
105           return user;
106       }
107
108       @Override
          public boolean insert(Object obj) {
110           User user = (User) obj;
111           try {
112               String toUserTable = "INSERT INTO tbl_user(firstName, lastName, email, ↩
    location, password) VALUES(?,?,?,?,?)";
113
114               //Stmt with returnable auto-generated key
                  PreparedStatement stmt = conn.prepareStatement(toUserTable, Statement.↩
    RETURN_GENERATED_KEYS);
116               stmt.setString(1, user.getFirstName());
117               stmt.setString(2, user.getLastName());
118               stmt.setString(3, user.getEmail());
119               stmt.setString(4, user.getLocation());
120               stmt.setString(5, user.getPassword());
121               stmt.executeUpdate();
122               ResultSet rs = stmt.getGeneratedKeys();
123               rs.next();
124               int userID = rs.getInt(1);//Return auto-generataed key to set ID of object
125               user.setId(userID);
126               stmt.close();
127               return true;
128           } catch (SQLException ex) {
129               JOptionPane.showMessageDialog(null, ex.getStackTrace(), ex.getMessage(), ↩
```

```
JOptionPane.ERROR_MESSAGE);
        }
        return false;
    }

    @Override
    public boolean update(Object obj) {
        String sql = "UPDATE tbl_user SET firstName = ?, lastName = ?, email = ?, location = ?, password = ? WHERE id = ?";
        User user = (User) obj;
        try {
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setString(1, user.getFirstName());
            stmt.setString(2, user.getLastName());
            stmt.setString(3, user.getEmail());
            stmt.setString(4, user.getLocation());
            stmt.setString(5, user.getPassword());
            stmt.setInt(6, user.getId());

            int rowsUpdated = stmt.executeUpdate();
            stmt.close();

            if (rowsUpdated > 0) {
                return true;
            }

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getStackTrace(), ex.getMessage(), JOptionPane.ERROR_MESSAGE);
        }
        return false;
    }

    @Override
    public boolean delete(Object obj) {
        String sql = "DELETE FROM tbl_user WHERE id = ?";
        User user = (User) obj;
        try {
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setInt(1, user.getId());

            int rowsDeleted = stmt.executeUpdate();
            stmt.close();
```

```
170
171                 if (rowsDeleted > 0) {
172                     return true;
173                 }
174
175             } catch (SQLException ex) {
176                 JOptionPane.showMessageDialog(null, ex.getStackTrace(), ex.getMessage(), ↩
    JOptionPane.ERROR_MESSAGE);
177             }
178             return false;
179         }
180
181         @Override
            public ArrayList<Object> getAll(int id) {
183             throw new UnsupportedOperationException("Not supported yet."); //To change ↩
    body of generated methods, choose Tools | Templates.
184         }
185
186     }
187
```

DAOUserTimeSlot.java

```java
package dao;

import dbutil.ConnectionFactory;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import javax.swing.JOptionPane;
import model.TimeSlot;
import model.User;

/**
 *
 * @author Biju Ale
 */
public class DAOUserTimeSlot implements DAO {

    Connection conn;

    public DAOUserTimeSlot() {
        conn = ConnectionFactory.getConnection();
    }

    @Override
    public ArrayList<Object> getAll() {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }

    @Override
    public Object get(int id) {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }

    @Override
    public boolean insert(Object obj) {
        User user = (User) obj;
        String toUserTimeSlotTable = "INSERT INTO tbl_user_timeslot(user_id, day_id, timeslot_type_id, timeslot_hr_block) VALUES(?,?,?,?)";
        //Stmt with returnable auto-generated key
        try {
```

```java
            PreparedStatement stmt = conn.prepareStatement(toUserTimeSlotTable, Statement.RETURN_GENERATED_KEYS);
            //Persisting all available timeslots
            for (TimeSlot availableTimeSlot : user.getAvailableTimeSlots()) {
                stmt.setInt(1, user.getId());
                stmt.setInt(2, availableTimeSlot.getDay());
                stmt.setInt(3, 1);
                stmt.setInt(4, availableTimeSlot.getHour());
                stmt.executeUpdate();
            }
            //Persisting all unavailable timeslots
            for (TimeSlot unavailableTimeSlot : user.getUnavailableTimeSlots()) {
                stmt.setInt(1, user.getId());
                stmt.setInt(2, unavailableTimeSlot.getDay());
                stmt.setInt(3, 2);
                stmt.setInt(4, unavailableTimeSlot.getHour());
                stmt.executeUpdate();
            }
            stmt.close();
            return true;
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getStackTrace(), ex.getMessage(), JOptionPane.ERROR_MESSAGE);
        }
        return false;
    }

    @Override
    public boolean update(Object obj) {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }

    @Override
    public boolean delete(Object obj) {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }

    @Override
    public ArrayList<Object> getAll(int id) {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }

}
```

SimpleDAOFactory.java

```java
package dao;

/**
 *
 * @author Biju Ale
 */
public class SimpleDAOFactory {

    static DAO daoUser, daoEvent, daoUserTimeSlot, daoInviteeEvent;

    public static DAO getDAO(String DAOType) {

        //Singleton pattern applied to DAO instantiations
        switch (DAOType) {
            case "DAOUser":
                if (daoUser == null) {
                    return new DAOUser();
                } else {
                    return daoUser;
                }
            case "DAOEvent":
                if (daoEvent == null) {
                    return new DAOEvent();
                } else {
                    return daoEvent;
                }
            case "DAOUserTimeSlot":
                if (daoUserTimeSlot == null) {
                    return new DAOUserTimeSlot();
                } else {
                    return daoUserTimeSlot;
                }
            case "DAOInviteeEvent":
                if (daoInviteeEvent == null) {
                    return new DAOInviteeEvent();
                } else {
                    return daoInviteeEvent;
                }
        }
        return null;
    }

}
```

Package – dbutil

ConnectionFactory.java

```java
package dbutil;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import javax.swing.JOptionPane;

/**
 *
 * @author Biju Ale
 */
public class ConnectionFactory {

    public static final String URL = "jdbc:mysql://127.0.0.1:3306/db_socialtime";
    public static final String USERNAME = "root";
    public static final String PASSWORD = "";

    private static Connection conn;

    private ConnectionFactory() {
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException ex) {
            JOptionPane.showMessageDialog(null, ex.getStackTrace(), ex.getMessage(), JOptionPane.ERROR_MESSAGE);
        }
    }

    public static Connection getConnection() {
        if (conn == null) {
            try {
                conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
            } catch (SQLException ex) {
                JOptionPane.showMessageDialog(null, ex.getStackTrace(), ex.getMessage(), JOptionPane.ERROR_MESSAGE);
            }
        }
        return conn;
    }

    public static void closeConnection() {
        if (conn != null) {
```

Package – model

CandidateTimeslot.java

```java
package model;

/**
 *
 * @author Biju Ale
 */
public class CandidateTimeslot {

    private TimeSlot timeSlot;
    private int voteCount;

    public TimeSlot getTimeSlot() {
        return this.timeSlot;
    }

    public void setTs(TimeSlot ts) {
        this.timeSlot = ts;
    }

    public int getVoteCount() {
        return voteCount;
    }

    public void setVoteCount(int voteCount) {
        this.voteCount = voteCount;
    }
}
```

Event.java

```java
package model;

import java.util.ArrayList;
import java.util.Date;
import subject_observer.Observer;
import subject_observer.Subject;

/**
 *
 * @author Biju Ale
 */
public class Event implements Subject {

    private int id;
    private User creator;
    private String name;
    private String location, description;
    private int duration;
    private Date date;
    private int minNoOfAttendees;
    private int minThresholdPercent;
    private ArrayList<User> invitee;
    private ArrayList<User> attendees;



    @Override
    public boolean registerObserver(Observer O) {
        invitee.add((User) O);
        return true;
    }

    @Override
    public boolean removeObserver(Observer O) {
        invitee.remove((User) O);
        return true;

    }

    @Override
    public boolean notifyObserver(Observer O) {
        for (User eachObserver : invitee) {
```

```
            eachObserver.notify();
        }
        return true;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public User getCreator() {
        return creator;
    }

    public void setCreator(User creator) {
        this.creator = creator;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getLocation() {
        return location;
    }

    public void setLocation(String location) {
        this.location = location;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
```

```java
    }

    public int getDuration() {
        return duration;
    }

    public void setDuration(int duration) {
        this.duration = duration;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public int getMinNoOfAttendees() {
        return minNoOfAttendees;
    }

    public void setMinNoOfAttendees(int minNoOfAttendees) {
        this.minNoOfAttendees = minNoOfAttendees;
    }

    public int getMinThresholdPercent() {
        return minThresholdPercent;
    }

    public void setMinThresholdPercent(int minThresholdPercent) {
        this.minThresholdPercent = minThresholdPercent;
    }

    public ArrayList<User> getInvitee() {
        return invitee;
    }

    public void setInvitee(ArrayList<User> invitee) {
        this.invitee = invitee;
    }

    public ArrayList<User> getAttendees() {
```

```
134         return attendees;
135     }
136
137     public void setAttendees(ArrayList<User> attendees) {
138         this.attendees = attendees;
139     }
140 }
141
```

Invitee.java

```java
package model;

/**
 *
 * @author Biju Ale
 */
public class Invitee extends User {

    public boolean viewEventList() {
        return false;
    }

    public boolean acceptInvitation() {
        return false;

    }

    public boolean rejectInvitation() {
        return false;

    }

    public boolean voteTimeSlot() {
        return false;

    }
}

```

TimeSlot.java

```java
package model;

/**
 *
 * @author Biju Ale
 */
public class TimeSlot {

    final private int day;
    final private int hour;

    public TimeSlot(int day, int hour) {
        this.day = day;
        this.hour = hour;
    }

    public int getDay() {
        return day;
    }

    public int getHour() {
        return hour;
    }

    @Override
    public String toString() {
        return Integer.toString(day) + " " + Integer.toString(hour);
    }
}
```

User.java

```java
package model;

import java.util.ArrayList;
import java.util.HashMap;
import subject_observer.EventObserver;

/**
 *
 * @author Biju Ale
 */
public class User implements EventObserver {

    private int id;
    private String firstName, lastName, email;
    private String location;
    private String password;
    ArrayList<Event> createdEvents;
    private ArrayList<TimeSlot> availableTimeSlots;
    private ArrayList<TimeSlot> unavailableTimeSlots;
    HashMap<String, String> notifications;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
```

```java
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getLocation() {
        return location;
    }

    public void setLocation(String location) {
        this.location = location;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public ArrayList<TimeSlot> getAvailableTimeSlots() {
        return availableTimeSlots;
    }

    public void setAvailableTimeSlots(ArrayList<TimeSlot> availableTimeSlots) {
        this.availableTimeSlots = availableTimeSlots;
    }

    public ArrayList<TimeSlot> getUnavailableTimeSlots() {
        return unavailableTimeSlots;
    }

    public void setUnavailableTimeSlots(ArrayList<TimeSlot> unavailableTimeSlots) {
        this.unavailableTimeSlots = unavailableTimeSlots;
    }

```

```
92          for (int i = 0; i < notifications.size(); i++) {
93
94          }
95      }
96      @Override
        public void showNotification(Event event) {
98          notifications.put("Event Name", email);
99      }
100
101     @Override
        public void showCandidateTimeSlots(Event event) {
103         throw new UnsupportedOperationException("Not supported yet."); //To change ↩
    body of generated methods, choose Tools | Templates.
104     }
105
106 }
107
```

Package – model.factory

```java
package model.factory;

import java.util.ArrayList;
import model.Event;
import model.User;

/**
 *
 * @author Biju Ale
 */
public class EventFactory {

    //for adding new event
    public static Event getModel(User creator, String eventName, String location, String description, int duration, int minAttendees, int minAThreshold, ArrayList<User> invitees) {
        Event event = new Event();
        event.setCreator(creator);
        event.setName(eventName);
        event.setLocation(location);
        event.setDescription(description);
        event.setDuration(duration);

        event.setMinNoOfAttendees(minAttendees);
        event.setMinThresholdPercent(minAThreshold);
        event.setInvitee(invitees);
        return event;
    }

    //for updating exsiting event
    public static Event getModel(int eventID, User creator, String eventName, String location, String description, int duration, int minAttendees, int minAThreshold) {
        Event event = new Event();
        event.setId(eventID);
        event.setCreator(creator);
        event.setName(eventName);
        event.setLocation(location);
        event.setDescription(description);
        event.setDuration(duration);
        event.setMinNoOfAttendees(minAttendees);
        event.setMinThresholdPercent(minAThreshold);
        return event;
    }
}
```

EventFactory.java

```java
package model.factory;

import java.util.ArrayList;
import model.Event;
import model.User;

/**
 *
 * @author Biju Ale
 */
public class EventFactory {

    //for adding new event
    public static Event getModel(User creator, String eventName, String location, String description, int duration, int minAttendees, int minAThreshold, ArrayList<User> invitees) {
        Event event = new Event();
        event.setCreator(creator);
        event.setName(eventName);
        event.setLocation(location);
        event.setDescription(description);
        event.setDuration(duration);

        event.setMinNoOfAttendees(minAttendees);
        event.setMinThresholdPercent(minAThreshold);
        event.setInvitee(invitees);
        return event;
    }

    //for updating exsiting event
    public static Event getModel(int eventID, User creator, String eventName, String location, String description, int duration, int minAttendees, int minAThreshold) {
        Event event = new Event();
        event.setId(eventID);
        event.setCreator(creator);
        event.setName(eventName);
        event.setLocation(location);
        event.setDescription(description);
        event.setDuration(duration);
        event.setMinNoOfAttendees(minAttendees);
        event.setMinThresholdPercent(minAThreshold);
        return event;
    }
```

```
46      }
47
```

TimeSlotFactory.java

```java
package model.factory;

import java.util.HashMap;
import java.util.Map;
import model.TimeSlot;

/**
 *
 * @author Biju Ale
 */
public class TimeSlotFactory {

    public static Map<String, TimeSlot> allPossibleTimeslots = new HashMap<>();
    TimeSlot timeslot;

    public static TimeSlot getTimeSlot(int day, int hour) {
        String key = Integer.toString(day) + Integer.toString(hour);
        if (allPossibleTimeslots.containsKey(key)) {
            return allPossibleTimeslots.get(key);
        }

        TimeSlot timeslot = new TimeSlot(day, hour);
        allPossibleTimeslots.put(key, timeslot);
        return timeslot;
    }

}
```

UserFactory.java

```java
package model.factory;

import java.util.ArrayList;
import model.TimeSlot;
import model.User;

/**
 *
 * @author Biju Ale
 */
public class UserFactory {

    public static User getModel(String firstName, String lastName, String email, String location, String password, ArrayList<TimeSlot> allAvailableTS, ArrayList<TimeSlot> allUnavailableTS) {
        User user = new User();
        user.setFirstName(firstName);
        user.setLastName(lastName);
        user.setEmail(email);
        user.setLocation(location);
        user.setPassword(password);
        user.setAvailableTimeSlots(allAvailableTS);
        user.setUnavailableTimeSlots(allUnavailableTS);
        return user;
    }

}
```

Package – subject_observer

EventObserver.java

```java
package subject_observer;

import model.Event;

/**
 *
 * @author Biju Ale
 */
public interface EventObserver {

    void showNotification(Event event);
    void showCandidateTimeSlots(Event event);

}

```

Observer.java (interface)

```java
package subject_observer;

/**
 *
 * @author Biju Ale
 */
public interface Observer {
    boolean update();
}

```

Subject.java (interface)

```java
package subject_observer;

/**
 *
 * @author Biju Ale
 */
public interface Subject {

    boolean registerObserver(Observer O);
    boolean removeObserver(Observer O);
    boolean notifyObserver(Observer O);
}

```

Package – view

AddEventFrame.java

```java
package view;

import controller.SocialTime;
import controller.EventController;
import controller.UserController;
import dao.DAOUser;
import dao.SimpleDAOFactory;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.HashSet;
import java.util.Set;
import java.util.Vector;
import javax.swing.BorderFactory;
import javax.swing.DefaultListCellRenderer;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JSpinner;
import javax.swing.SpinnerNumberModel;
import javax.swing.JTextArea;
import javax.swing.ListSelectionModel;
import javax.swing.border.TitledBorder;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import model.User;

public class AddEventFrame extends JFrame implements ActionListener, ↩
ListSelectionListener {

    private JPanel contentPane;
    private JTextField txtEventName, txtLocation;
```

```
43          private JTextArea txtDescription;
44          private JSpinner spnMinAttendee, spnMinThreshold, spnDuration;
45          JButton btnSubmit;
46
47          JPanel pnlLists;
48          private final JList lstAllUsers, lstToInvite;
49
50          Color foreGroundColor = new Color(102, 255, 204);
51          Color backgroundColor = new Color(102, 0, 51);
52          EventController eventController = SocialTime.getEventController();
53
54          Set<User> colLstAllUsers = new HashSet();
55          Set<User> colLstToInvite = new HashSet<>();
56
57          public AddEventFrame() {
            setTitle("Add event");
59      //        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            setBounds(100, 100, 420, 680);
            setLocationRelativeTo(null);
62          contentPane = new JPanel();
63          contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
            setContentPane(contentPane);
65          contentPane.setLayout(null);
66          contentPane.setBackground(new Color(102, 0, 51));
67
68          JLabel lblNewLabel = new JLabel("Event Name");
69          lblNewLabel.setBounds(82, 25, 85, 14);
70          lblNewLabel.setForeground(new Color(102, 255, 204));
71
72          contentPane.add(lblNewLabel);
73
74          JLabel lblLocation = new JLabel("Location");
75          lblLocation.setBounds(100, 50, 67, 14);
76          lblLocation.setForeground(new Color(102, 255, 204));
77          contentPane.add(lblLocation);
78
79          JLabel lblDescription = new JLabel("Description");
80          lblDescription.setBounds(87, 75, 80, 14);
81          lblDescription.setForeground(new Color(102, 255, 204));
82          contentPane.add(lblDescription);
83
84          JLabel lblMinAttendees = new JLabel("Minimum No. of attendees");
```

```
85          lblMinAttendees.setBounds(25, 181, 200, 14);
86          lblMinAttendees.setForeground(new Color(102, 255, 204));
87          contentPane.add(lblMinAttendees);
88
89          JLabel lblMinimumThresholdPercent = new JLabel("Minimum Threshold Percent");
90          lblMinimumThresholdPercent.setBounds(15, 208, 160, 14);
91          lblMinimumThresholdPercent.setForeground(new Color(102, 255, 204));
92          contentPane.add(lblMinimumThresholdPercent);
93
94  //        JLabel lblInvite = new JLabel("Select list item to transfer users between ↩
    lists");
95  //        lblInvite.setBounds(45, 240, 300, 14);
96  //        lblInvite.setForeground(new Color(110, 255, 204));
97  //        contentPane.add(lblInvite);
98          txtEventName = new JTextField();
99          txtEventName.setBounds(177, 25, 200, 20);
100         contentPane.add(txtEventName);
101         txtEventName.setColumns(10);
102
103         txtLocation = new JTextField();
104         txtLocation.setColumns(10);
105         txtLocation.setBounds(177, 50, 200, 20);
106         contentPane.add(txtLocation);
107
108         btnSubmit = new JButton("Submit");
109         btnSubmit.setBounds(300, 600, 89, 23);
            btnSubmit.addActionListener(this);
111         contentPane.add(btnSubmit);
112
113         spnMinAttendee = new JSpinner();
114         spnMinAttendee.setModel(new SpinnerNumberModel(0, 0, 100, 1));
115         spnMinAttendee.setBounds(200, 178, 55, 20);
116         contentPane.add(spnMinAttendee);
117
118         spnMinThreshold = new JSpinner();
119         spnMinThreshold.setModel(new SpinnerNumberModel(0, 0, 100, 1));
120         spnMinThreshold.setBounds(200, 205, 55, 20);
121         contentPane.add(spnMinThreshold);
122
123         txtDescription = new JTextArea();
124         txtDescription.setLineWrap(true);
125         txtDescription.setBounds(177, 78, 200, 60);
```

```
126         contentPane.add(txtDescription);
127
128         spnDuration = new JSpinner();
129         spnDuration.setModel(new SpinnerNumberModel(0, 0, 100, 1));
130         spnDuration.setBounds(177, 150, 55, 20);
131
132         //Fetch all users as potential invitee
            for (Object each : ((DAOUser) SimpleDAOFactory.getDAO("DAOUser")).getAll()) {
134             User user = (User) each;
135             colLstAllUsers.add(user);
136             System.out.println(user.getId());
137             System.out.println(user.getFirstName());
138         }
139         //Remove logged in user from potential invitee list
140         for (int i = 0; i < colLstAllUsers.size(); i++) {
141             colLstAllUsers.remove(UserController.getLoggedInUser());
142
143         }
144
145         //Prepare JList for potential invitee
            lstAllUsers = new JList(new Vector<User>(colLstAllUsers));
            lstAllUsers.addListSelectionListener(this);
148         lstAllUsers.setPreferredSize(new Dimension(150, 70));
149         lstAllUsers.setBorder(BorderFactory.createTitledBorder(BorderFactory.↩
    createEtchedBorder(this.backgroundColor, this.foreGroundColor), "Potential Invitee", ↩
    TitledBorder.DEFAULT_JUSTIFICATION, TitledBorder.TOP, new Font("Georgia", Font.PLAIN, ↩
    14), backgroundColor));
150         lstAllUsers.setVisibleRowCount(10);
151         lstAllUsers.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
152         lstAllUsers.setCellRenderer(new DefaultListCellRenderer() {
153             @Override//Display Object's data member - string(User's name in list)
                public Component getListCellRendererComponent(JList<?> list, Object value, ↩
    int index, boolean isSelected, boolean cellHasFocus
155             ) {
156                 Component renderer = super.getListCellRendererComponent(list, value, ↩
    index, isSelected, cellHasFocus);
157                 if (renderer instanceof JLabel && value instanceof User) {
158                     // Here value will be of the Type 'User'
159                     ((JLabel) renderer).setText(((User) value).getFirstName() + " " + (↩
    (User) value).getLastName());
160                 }
161                 return renderer;
```

```
162                 }
163             });
164
165             //Prepare JList for invited users
166             colLstToInvite.add(null);
                lstToInvite = new JList(new Vector<User>(colLstToInvite));
                lstToInvite.addListSelectionListener(this);
169             lstToInvite.setPreferredSize(new Dimension(150, 70));
170             lstToInvite.setBorder(BorderFactory.createTitledBorder(BorderFactory.↩
    createEtchedBorder(this.backgroundColor, this.foreGroundColor), "To invite", ↩
    TitledBorder.DEFAULT_JUSTIFICATION, TitledBorder.TOP, new Font("Georgia", Font.PLAIN, ↩
    14), backgroundColor));
171             lstToInvite.setVisibleRowCount(10);
172             lstToInvite.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
173             lstToInvite.setCellRenderer(new DefaultListCellRenderer() {
174                 @Override
                    public Component getListCellRendererComponent(JList<?> list, Object value, ↩
    int index, boolean isSelected, boolean cellHasFocus
176                 ) {
177                     Component renderer = super.getListCellRendererComponent(list, value, ↩
    index, isSelected, cellHasFocus);
178                     if (renderer instanceof JLabel && value instanceof User) {
179                         // Here value will be of the Type 'User'
180                         ((JLabel) renderer).setText(((User) value).getFirstName() + " " + (↩
    (User) value).getLastName());
181                     }
182                     return renderer;
183                 }
184             });
185
186             pnlLists = new JPanel(new BorderLayout());
187             pnlLists.setBorder(BorderFactory.createTitledBorder(BorderFactory.↩
    createEtchedBorder(this.backgroundColor, this.foreGroundColor), "Select available ↩
    timeslots (Recurrs weekly)", TitledBorder.DEFAULT_JUSTIFICATION, TitledBorder.TOP, new ↩
    Font("Georgia", Font.PLAIN, 14), new Color(102, 255, 204)));
188             pnlLists.setOpaque(false);
189             pnlLists.add(new JScrollPane(lstAllUsers), BorderLayout.WEST);
190             pnlLists.add(new JScrollPane(lstToInvite), BorderLayout.EAST);
191             pnlLists.setBounds(2, 250, 400, 330);
192             contentPane.add(pnlLists);
                setVisible(true);
194         }
```

```java
    @Override
    public void actionPerformed(ActionEvent e) {
        Object btnSource = e.getSource();
        if (btnSource.equals(btnSubmit)) {

            //Null validation
            if (txtEventName.getText().equals(null) || txtEventName.getText().equals("")) {
                JOptionPane.showMessageDialog(null, "Event name cannot be left blank.");
                txtEventName.setText("");
                txtEventName.requestFocus();
                return;
            }
            if (txtLocation.getText().equals(null) || txtLocation.getText().equals("")) {
                JOptionPane.showMessageDialog(null, "Location cannot be left blank!");
                txtLocation.setText("");
                txtLocation.requestFocus();
                return;
            }

            if (txtDescription.getText().equals(null) || txtLocation.getText().equals("")) {
                JOptionPane.showMessageDialog(null, "Description cannot be left blank!");
                txtDescription.setText("");
                txtDescription.requestFocus();
                return;
            }

            String eventName = txtEventName.getText();
            String location = txtLocation.getText();
            String description = txtDescription.getText();
            int minAttendees = (Integer) spnMinAttendee.getValue();
            int minAThreshold = (Integer) spnMinThreshold.getValue();
            int duration = (Integer) spnDuration.getValue();
            eventController.saveEvent(eventName, location, description, duration, minAttendees, minAThreshold, colLstToInvite);
            JOptionPane.showMessageDialog(null, "Event saved Succesfully!", "Event has been added!", JOptionPane.INFORMATION_MESSAGE);
        }
```

```java
    public void resetFields() {
        //Clear all fields
        txtEventName.setText("");
        txtLocation.setText("");
        txtDescription.setText("");
        spnMinAttendee.setValue(1);
        spnMinThreshold.setValue(1);
        spnDuration.setValue(1);
    }

    @Override
    public void valueChanged(ListSelectionEvent e) {
        Object lstSource = e.getSource();

        if (lstSource.equals(lstAllUsers)) {
            User selectedUser = ((User) lstAllUsers.getSelectedValue());
            colLstAllUsers.remove(selectedUser);
            colLstToInvite.add(selectedUser);
            lstAllUsers.setListData(new Vector<User>(colLstAllUsers));
            lstToInvite.setListData(new Vector<User>(colLstToInvite));
        } else if (lstSource.equals(lstToInvite)) {
            User selectedUser = ((User) lstToInvite.getSelectedValue());
            colLstToInvite.remove(selectedUser);
            colLstAllUsers.add(selectedUser);
            lstAllUsers.setListData(new Vector<User>(colLstAllUsers));
            lstToInvite.setListData(new Vector<User>(colLstToInvite));
        }
    }
}
```

Dashboard.java

```java
package view;

import controller.EventController;
import controller.UserController;
import dao.SimpleDAOFactory;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Arrays;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.UIManager;
import static javax.swing.WindowConstants.DISPOSE_ON_CLOSE;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import model.Event;
import subject_observer.Observer;

/**
 *
 * @author Biju Ale
 */
public class Dashboard extends JFrame implements ActionListener, Observer {

    static Dashboard dashboard;

    JTable tblCreatedEvents, tblInvitedEvents;
    TableModel tableModel;
    JScrollPane scrollPane;
    JPanel pnlTable, pnlTable2;
    private final JButton btnLogout;
    private final JButton btnEditProfile;
```

```
49      private final JButton btnCreateEvent;
50      private final JPanel pnlWest;
51      private final JButton btnEditEvent;
52
53      EventController eventController = new EventController();
54      private final JButton btnViewInvitations;
55
56      Color foreGroundColor = new Color(102, 255, 204);
57      Color backgroundColor = new Color(102, 0, 51);
58
59      public static Dashboard getInstanceOf() {
60          if (dashboard == null) {
61              dashboard = new Dashboard();
62              dashboard.setVisible(true);
63          }
64          dashboard.setVisible(true);
65          return dashboard;
66      }
67
68      private Dashboard() {
69          Font f = new javax.swing.plaf.FontUIResource("Georgia", Font.PLAIN, 12);
70          java.util.Enumeration keys = UIManager.getDefaults().keys();
71          while (keys.hasMoreElements()) {
72              Object key = keys.nextElement();
73              Object value = UIManager.get(key);
74              if (value != null && value instanceof javax.swing.plaf.FontUIResource) {
75                  UIManager.put(key, f);
76              }
77          }
78
            setTitle("Dashboard - Social Time");
            setLayout(new BorderLayout());
            setDefaultCloseOperation(DISPOSE_ON_CLOSE);
            getContentPane().setBackground(backgroundColor);
83
84          pnlWest = new JPanel(new FlowLayout());
85          pnlWest.setOpaque(false);
86          pnlWest.setPreferredSize(new Dimension(150, 20));
87
88          JLabel lblFullName = new JLabel(UserController.getLoggedInUser().getFirstName()↩
    + " " + UserController.getLoggedInUser().getLastName());
89          lblFullName.setFont(new Font("Georgia", Font.PLAIN, 11));
90          lblFullName.setForeground(foreGroundColor);
```

```
91          pnlWest.add(lblFullName, BorderLayout.WEST);
92
93          JLabel lblDashboard = new JLabel("DASHBOARD");
94          lblDashboard.setFont(new Font("Georgia", Font.PLAIN, 20));
95          lblDashboard.setForeground(foreGroundColor);
96          pnlWest.add(lblDashboard);
97
98          btnLogout = new JButton("Logout");
            btnLogout.addActionListener(this);
100         pnlWest.add(btnLogout, BorderLayout.WEST);
101 //        pnlWest.add(btnLogout);
102
103         btnEditProfile = new JButton("Edit Profile");
            btnEditProfile.addActionListener(this);
105         pnlWest.add(btnEditProfile);
106
107         btnCreateEvent = new JButton("Create Event");
            btnCreateEvent.addActionListener(this);
109         pnlWest.add(btnCreateEvent);
110
111         btnEditEvent = new JButton("Edit Event");
            btnEditEvent.addActionListener(this);
113         pnlWest.add(btnEditEvent);
114
115         btnViewInvitations = new JButton("View Invitations");
            btnViewInvitations.addActionListener(this);
117         pnlWest.add(btnViewInvitations);
118
119         pnlTable = new JPanel();
120         pnlTable2 = new JPanel();
121         //Blank tblCreatedEvents instantiation
122         String[] columnHeadings = {"EVENT ID", "NAME", "DESCRIPTION", "DURATION", "↩
    LOCATION", "MIN. ATTENDEES", "MIN THRESHOLD"};
123         tableModel = new DefaultTableModel(columnHeadings, 0);
124         tblCreatedEvents = new JTable(tableModel) {//Creating tblCreatedEvents object &↩
     making tblCreatedEvents anonymous class to make cell editable false.
125             @Override
                public boolean isCellEditable(int data, int column) {
127                 return false;
128             }
129         };
130
```

```
131         //Inserting vertical scrollbar to tblCreatedEvents
132         tblCreatedEvents.setAutoCreateRowSorter(true);
133         tblCreatedEvents.setPreferredScrollableViewportSize(new Dimension(890, 400));
134         tblCreatedEvents.setFillsViewportHeight(true);
135         scrollPane = new JScrollPane(tblCreatedEvents);
136         pnlTable.add(scrollPane);
137
            add(pnlWest, BorderLayout.WEST);
            add(pnlTable, BorderLayout.CENTER);
140
            pack();
            setVisible(true);
            setLocationRelativeTo(null);
144     }
145
146     public void updateTable(ArrayList<Object> events) throws SQLException {
147         DefaultTableModel model = (DefaultTableModel) tblCreatedEvents.getModel();
148         model.setRowCount(0);
149         String[] tuple = new String[7];
            for (Object event : events) { //Loop through all indices/rooms in this stack
151             Event eachEvent = (Event) event;
152             tuple[0] = Integer.toString(eachEvent.getId()); //Using getters of each ↩
    transaction to access their state/field.
153             tuple[1] = eachEvent.getName();
154             tuple[2] = eachEvent.getDescription();
155             tuple[3] = Integer.toString(eachEvent.getDuration());
156             tuple[4] = eachEvent.getLocation();
157             tuple[5] = Integer.toString(eachEvent.getMinNoOfAttendees());
158             tuple[6] = Integer.toString(eachEvent.getMinThresholdPercent());
159             model.addRow(tuple);
160             Arrays.fill(tuple, null); //Clearing all elements from tuple for next ↩
    transaction data
161         }
162 //          tblCreatedEvents.repaint();
163
164     }
165
166     @Override
        public void actionPerformed(ActionEvent e) {
168
169         int selectedRowNo = tblCreatedEvents.getSelectedRow();
170
```

```
171         Object btnSource = (Object) e.getSource();
172         if (btnSource.equals(btnCreateEvent)) {
173             GUIFactory.getInstanceOf("addEventFrame");
174         } else if (btnSource.equals(btnEditEvent)) {
175             if (selectedRowNo != -1) {//If row is selected (selected row no. is greater↩
 than -1)
176                 UpdateEventFrame updateEventFrame = (UpdateEventFrame) GUIFactory.↩
getInstanceOf("updateEventFrame");
177                 updateEventFrame.setVisible(true);
178 
179                 int eventID = Integer.parseInt((String) tblCreatedEvents.getValueAt(↩
selectedRowNo, 0));
180                 updateEventFrame.setEventID(eventID);
181 
182                 String eventName = (String) tblCreatedEvents.getValueAt(selectedRowNo, ↩
1);
183                 updateEventFrame.setTxtEventName(eventName);
184 
185                 String description = (String) tblCreatedEvents.getValueAt(selectedRowNo↩
, 2);
186                 updateEventFrame.setTxtDescription(description);
187 
188                 int duration = Integer.parseInt((String) tblCreatedEvents.getValueAt(↩
selectedRowNo, 3));
189                 updateEventFrame.setSpnDuration(duration);
190 
191                 String location = (String) tblCreatedEvents.getValueAt(selectedRowNo, 4↩
);
192                 updateEventFrame.setTxtLocation(location);
193 
194                 int minAttendee = Integer.parseInt((String) tblCreatedEvents.getValueAt↩
(selectedRowNo, 5));
195                 updateEventFrame.setSpnMinAttendee(minAttendee);
196 
197                 int minThreshold = Integer.parseInt((String) tblCreatedEvents.↩
getValueAt(selectedRowNo, 6));
198                 updateEventFrame.setSpnMinThreshold(minThreshold);
199 
200             } else {
201                 JOptionPane.showMessageDialog(null, "No cell selected", "Select event ↩
from table to update!", JOptionPane.WARNING_MESSAGE);
202             }
```

```
203             } else if (btnSource.equals(btnEditProfile)) {
204                 GUIFactory.getInstanceOf("updateProfileFrame");
205             } else if (btnSource.equals(btnLogout)) {
206                 this.dispose();
207                 GUIFactory.getInstanceOf("loginFrame");
208 
209             } else if (btnSource.equals(btnViewInvitations)) {
210                 SimpleDAOFactory.getDAO("DAOInviteeEvent").getAll();
211                 GUIFactory.getInstanceOf("invitationsFrame");
212 
213             }
214         }
215 
216         @Override
    ⓘ       public boolean update() {
218             throw new UnsupportedOperationException("Not supported yet."); //To change body↩
    of generated methods, choose Tools | Templates.
219         }
220     }
221 
```

GUIFactory.java

```java
package view;

/**
 *
 * @author Biju Ale
 */
public abstract class GUIFactory {

    private static LoginFrame loginFrame;
    private static RegistrationFrame regisrationFrame;
    private static AddEventFrame addEventFrame;
    private static Dashboard dashboard;
    private static UpdateEventFrame updateEventFrame;
    private static UpdateProfileFrame updateProfileFrame;
    private static InvitationsFrame invitationsFrame;

    //Singleton instantiation of GUI elements
    public static Object getInstanceOf(String GUIType) {
        switch (GUIType) {
            case "loginFrame":
                if (loginFrame == null) {
                    return new LoginFrame();
                } else {
                    return loginFrame;
                }
            case "regisrationFrame":
                if (regisrationFrame == null) {
                    return new RegistrationFrame();
                } else {
                    return regisrationFrame;
                }
            case "dashboard":
                if (dashboard == null) {
                    dashboard = Dashboard.getInstanceOf();
                } else {
                    return dashboard;
                }
            case "addEventFrame":
                if (addEventFrame == null) {
                    return new AddEventFrame();
                } else {
                    return addEventFrame;
```

```
48                  }
49              case "updateEventFrame":
50                  if (updateEventFrame == null) {
51                      return new UpdateEventFrame();
52                  } else {
53                      return updateEventFrame;
54                  }
55              case "updateProfileFrame":
56                  if (updateProfileFrame == null) {
57                      return new UpdateProfileFrame();
58                  } else {
59                      return updateProfileFrame;
60                  }
61              case "invitationsFrame":
62                  if (invitationsFrame == null) {
63                      invitationsFrame = InvitationsFrame.getInstanceOf();
64                  } else {
65                      return invitationsFrame;
66                  }
67          }
68          return null;
69      }
70  }
71
```

InvitationsFrame.java

```java
package view;

import controller.EventController;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Arrays;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.UIManager;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import model.Event;
import subject_observer.Observer;

/**
 *
 * @author Biju Ale
 */
public class InvitationsFrame extends JFrame implements ActionListener, Observer {

    static InvitationsFrame invitationsFrame;

    JTable tblInvitations, tblInvitedEvents;
    TableModel tableModel;
    JScrollPane scrollPane;
    JPanel pnlTable, pnlTable2;
    private final JPanel pnlWest;

    EventController eventController = new EventController();

    Color foreGroundColor = new Color(102, 255, 204);
    Color backgroundColor = new Color(102, 0, 51);
```

```
49
50      public static InvitationsFrame getInstanceOf() {
51          if (invitationsFrame == null) {
52              invitationsFrame = new InvitationsFrame();
53              invitationsFrame.setVisible(true);
54          }
55          invitationsFrame.setVisible(true);
56          return invitationsFrame;
57      }
58      private final JButton btnAccept, btnDecline;
59      private final JButton btnDashboard;
60
61      private InvitationsFrame() {
62          Font f = new javax.swing.plaf.FontUIResource("Georgia", Font.PLAIN, 12);
63          java.util.Enumeration keys = UIManager.getDefaults().keys();
64          while (keys.hasMoreElements()) {
65              Object key = keys.nextElement();
66              Object value = UIManager.get(key);
67              if (value != null && value instanceof javax.swing.plaf.FontUIResource) {
68                  UIManager.put(key, f);
69              }
70          }
71
            setTitle("Dashboard - Social Time");
            setLayout(new BorderLayout());
74  //        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
            getContentPane().setBackground(backgroundColor);
76
77          pnlWest = new JPanel(new FlowLayout());
78          pnlWest.setOpaque(false);
79          pnlWest.setPreferredSize(new Dimension(150, 20));
80
81          JLabel lblTitle = new JLabel("Invitations");
82          lblTitle.setFont(new Font("Georgia", Font.PLAIN, 20));
83          lblTitle.setForeground(foreGroundColor);
84          pnlWest.add(lblTitle);
85
86          btnDashboard = new JButton("Dashboard");
            btnDashboard.addActionListener(this);
88          pnlWest.add(btnDashboard);
89
90          btnAccept = new JButton("Accept");
```

```java
        btnAccept.addActionListener(this);
        pnlWest.add(btnAccept);

        btnDecline = new JButton("Decline");
        btnDecline.addActionListener(this);
        pnlWest.add(btnDecline);

        pnlTable = new JPanel();
        pnlTable2 = new JPanel();
        //Blank tblCreatedEvents instantiation
        String[] columnHeadings = {"EVENT ID", "NAME", "DESCRIPTION", "DURATION", "LOCATION", "MIN. ATTENDEES", "MIN THRESHOLD"};
        tableModel = new DefaultTableModel(columnHeadings, 0);
        tblInvitations = new JTable(tableModel) {//Creating tblCreatedEvents object & making tblCreatedEvents anonymous class to make cell editable false.
            @Override
            public boolean isCellEditable(int data, int column) {
                return false;
            }
        };

        //Inserting vertical scrollbar to tblCreatedEvents
        tblInvitations.setAutoCreateRowSorter(true);
        tblInvitations.setPreferredScrollableViewportSize(new Dimension(890, 400));
        tblInvitations.setFillsViewportHeight(true);
        scrollPane = new JScrollPane(tblInvitations);
        pnlTable.add(scrollPane);

        add(pnlWest, BorderLayout.WEST);
        add(pnlTable, BorderLayout.CENTER);

        pack();
        setVisible(true);
        setLocationRelativeTo(null);
    }

    public void updateTable(ArrayList<Object> events) throws SQLException {
        DefaultTableModel model = (DefaultTableModel) tblInvitations.getModel();
        model.setRowCount(0);
        String[] tuple = new String[7];
        for (Object event : events) { //Loop through all indices/rooms in this stack
            Event eachEvent = (Event) event;
```

```
131                 tuple[0] = Integer.toString(eachEvent.getId()); //Using getters of each ↩
    transaction to access their state/field.
132                 tuple[1] = eachEvent.getName();
133                 tuple[2] = eachEvent.getDescription();
134                 tuple[3] = Integer.toString(eachEvent.getDuration());
135                 tuple[4] = eachEvent.getLocation();
136                 tuple[5] = Integer.toString(eachEvent.getMinNoOfAttendees());
137                 tuple[6] = Integer.toString(eachEvent.getMinThresholdPercent());
138                 model.addRow(tuple);
139                 Arrays.fill(tuple, null); //Clearing all elements from tuple for next ↩
    transaction data
140             }
141     //        tblCreatedEvents.repaint();
142
143         }
144
145         public static void main(String[] args) {
            new InvitationsFrame();
147         }
148
149         @Override
            public void actionPerformed(ActionEvent e) {
151             int selectedRowNo = tblInvitations.getSelectedRow();
152             Object btnSource = (Object) e.getSource();
153             if (btnSource.equals(btnDashboard)) {
154                 this.dispose();
155                 GUIFactory.getInstanceOf("dashboard");
156
157             }
158
159         }
160
161         @Override
            public boolean update() {
163             throw new UnsupportedOperationException("Not supported yet."); //To change body↩
     of generated methods, choose Tools | Templates.
164         }
165     }
166
```

LoginFrame.java

```java
package view;

import controller.SocialTime;
import controller.UserController;
import dao.DAO;
import dao.DAOEvent;
import dao.SimpleDAOFactory;
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JLabel;
import java.awt.Font;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.Color;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JOptionPane;
import javax.swing.JPasswordField;

public class LoginFrame extends JFrame implements ActionListener {

    private JTextField txtEmail;
    private JPasswordField txtPassword;
    JButton btnLogin, btnRegister;
    RegistrationFrame regFrame;
    UserController userController = SocialTime.getUserController();

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    LoginFrame frame = new LoginFrame();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
```

```
43                  }
44              }
45          });
46      }
47
48      public LoginFrame() {
        setSize(250, 225);
        setTitle("Social Time - Login");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        getContentPane().setBackground(new Color(102, 0, 51));
        getContentPane().setLayout(null);
54
55          btnLogin = new JButton("Login");
            btnLogin.addActionListener(this);
57          btnRegister = new JButton("Register");
            btnRegister.addActionListener(this);
59
60          JLabel lblNewLabel = new JLabel("Event Planner Tool by Biju Ale");
61          lblNewLabel.setForeground(new Color(102, 255, 204));
62          lblNewLabel.setFont(new Font("Georgia", Font.ITALIC, 11));
63          lblNewLabel.setBounds(21, 34, 163, 25);
            getContentPane().add(lblNewLabel);
65
66          JLabel label = new JLabel("Social Time");
67          label.setForeground(new Color(102, 255, 204));
68          label.setFont(new Font("Georgia", Font.PLAIN, 20));
69          label.setBounds(21, 11, 119, 25);
            getContentPane().add(label);
71
72          btnLogin.setFont(new Font("Georgia", Font.PLAIN, 11));
73
74          btnLogin.setBounds(21, 139, 89, 23);
            getContentPane().add(btnLogin);
76
77          btnRegister.setFont(new Font("Georgia", Font.PLAIN, 11));
78          btnRegister.setBounds(120, 139, 93, 23);
            getContentPane().add(btnRegister);
80
81          JLabel lblEmail = new JLabel("Email");
82          lblEmail.setForeground(new Color(102, 255, 204));
83          lblEmail.setFont(new Font("Georgia", Font.PLAIN, 11));
84          lblEmail.setBounds(21, 80, 44, 25);
```

```
        getContentPane().add(lblEmail);

        JLabel lblPassword = new JLabel("Password");
        lblPassword.setForeground(new Color(102, 255, 204));
        lblPassword.setFont(new Font("Georgia", Font.PLAIN, 11));
        lblPassword.setBounds(21, 106, 65, 25);
        getContentPane().add(lblPassword);

        txtEmail = new JTextField();
        txtEmail.setBounds(84, 82, 129, 20);
        getContentPane().add(txtEmail);
        txtEmail.setColumns(10);

        txtPassword = new JPasswordField();
        txtPassword.setColumns(10);
        txtPassword.setBounds(84, 108, 129, 20);
        getContentPane().add(txtPassword);

        setLocationRelativeTo(null);

        regFrame = new RegistrationFrame();
        regFrame.setVisible(false);
        //Anonymous class to override close operation. When form is closed it is hidden
        regFrame.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                regFrame.dispose();
            }
        });
        //Anonymous class to override show and hide operation. All fields are reset.
        regFrame.addComponentListener(new ComponentAdapter() {
            public void componentHidden(ComponentEvent e) {
                regFrame.resetFields();
            }

            public void componentShown(ComponentEvent e) {
                regFrame.resetFields();
            }
        });
        regFrame.loginFrame = this;
        txtEmail.requestFocus();
```

```java
        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        Object btnSource = e.getSource();
        if (btnSource.equals(btnRegister)) {
            regFrame.setVisible(true);
            regFrame.btnReset.doClick(); //Reset all fields
        } else if (btnSource.equals(btnLogin)) {

            //Null validation
            if (txtEmail.getText().equals(null) || txtEmail.getText().equals("")) {
                JOptionPane.showMessageDialog(null, "Email cannot be left blank!");
                txtEmail.setText("");
                txtEmail.requestFocus();
                return;
            }
            if (new String(txtPassword.getPassword()).equals(null) || new String(txtPassword.getPassword()).equals("")) {
                JOptionPane.showMessageDialog(null, "Password cannot be left blank!");
                txtPassword.setText("");
                txtPassword.requestFocus();
                return;
            }

            String email = txtEmail.getText();
            String password = new String(txtPassword.getPassword());
            if (userController.authenticateUser(email, password)) {
                JOptionPane.showMessageDialog(null, "Welcome!");
                Dashboard dashboard = Dashboard.getInstanceOf();
                DAO daoEvent = SimpleDAOFactory.getDAO("DAOEvent");
                DAOEvent.dashboard = dashboard;//Dashboard is registered for syncing created event list
                daoEvent.getAll(UserController.getLoggedInUser().getId());
                this.dispose();
                dashboard.setVisible(true);
            } else {
                JOptionPane.showMessageDialog(null, "Invalid username or password.");
                txtEmail.setText("");
                txtPassword.setText("");
                txtEmail.requestFocus();
                return;
            }
        }
    }
}
```

RegistrationFrame.java

```java
package view;

import controller.SocialTime;
import controller.TimeSlotController;
import controller.UserController;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.util.ArrayList;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import javax.swing.border.TitledBorder;
import model.TimeSlot;

/**
 *
 * @author Biju Ale
 */
public class RegistrationFrame extends JFrame implements ActionListener, ItemListener {

    private JCheckBox[][] allChkAvailableTS, allChkUnavailableTS;
    private JCheckBox eachChkAvailableTS, eachChkUnavailableTS;

    private final JTextField txtFirstName;
    private final JTextField txtLastName = new JTextField(20);
```

```
48      private final JTextField txtLocation = new JTextField(20);
49      private final JTextField txtEmail = new JTextField(20);
50      private final JPasswordField txtPassword = new JPasswordField(20);
51
52      private final JButton btnRegister = new JButton("Register");
53      final JButton btnReset = new JButton("Reset");
54      private final JButton btnLogin = new JButton("Back to Login");
55
56      LoginFrame loginFrame;
57
58      private final ArrayList<String> allAvailableTS_Str = new ArrayList();
59      private final ArrayList<String> allUnavailableTS_Str = new ArrayList();
60
61      private final UserController userController = SocialTime.getUserController();
62      private final TimeSlotController timeslotController = new TimeSlotController();
63
64      Color foreGroundColor = new Color(102, 255, 204);
65      Color backgroundColor = new Color(102, 0, 51);
66
67      JList<String> potentialInvitees, toInviteUsers;
68
69      public RegistrationFrame() {
70
71          this.txtFirstName = new JTextField(20);
            setTitle("Account Registration - Social Time");
            getContentPane().setBackground(backgroundColor);
            getContentPane().setLayout(new FlowLayout());
            setSize(720, 820);
            setLocationRelativeTo(null);
77
78          allChkAvailableTS = new JCheckBox[8][24];
79          allChkUnavailableTS = new JCheckBox[8][24];
80
81          JPanel topContainer = new JPanel();
82          topContainer.setPreferredSize(new Dimension(300, 150));
83          topContainer.setBackground(backgroundColor);
84
85          JLabel lblFirstName = new JLabel("First Name");
86          lblFirstName.setForeground(foreGroundColor);
87          topContainer.add(lblFirstName);
88          topContainer.add(txtFirstName);
89
90          JLabel lblLastName = new JLabel("Last Name");
```

```
91          lblLastName.setForeground(foreGroundColor);
92          topContainer.add(lblLastName);
93          topContainer.add(txtLastName);
94
95          JLabel lblLocation = new JLabel("Location");
96          lblLocation.setForeground(foreGroundColor);
97          topContainer.add(lblLocation);
98          topContainer.add(txtLocation);
99
100         JLabel lblEmail = new JLabel("Email");
101         lblEmail.setForeground(foreGroundColor);
102         topContainer.add(lblEmail);
103         topContainer.add(txtEmail);
104
105         JLabel lblPassword = new JLabel("Password");
106         lblPassword.setForeground(foreGroundColor);
107         topContainer.add(lblPassword);
108         topContainer.add(txtPassword);
109
            btnRegister.addActionListener(this);
            btnReset.addActionListener(this);
            btnLogin.addActionListener(this);
113
            add(topContainer);
            add(inputAreaAvailableTS());
            add(inputAreaUnavailableTS());
            add(btnRegister);
            add(btnReset);
            add(btnLogin);
120     }
121
122     private JPanel inputAreaAvailableTS() {
123         JPanel pnlAvailableTS = new JPanel(new FlowLayout());
124         pnlAvailableTS.setBackground(new Color(102, 0, 51));
125         pnlAvailableTS.setPreferredSize(new Dimension(670, 280));
126         pnlAvailableTS.setBorder(BorderFactory.createTitledBorder(BorderFactory.↩
    createEtchedBorder(this.backgroundColor, this.foreGroundColor), "Select available ↩
    timeslots (Recurrs weekly)", TitledBorder.DEFAULT_JUSTIFICATION, TitledBorder.TOP, new ↩
    Font("Georgia", Font.PLAIN, 14), new Color(102, 255, 204)));
127
128         pnlAvailableTS.add(new Canvas());//Hour labels
129         allChkAvailableTS = new JCheckBox[8][24];
```

```java
        for (int dayNo = 1; dayNo < 8; dayNo++) {
            for (int hour = 0; hour < 24; hour++) {
                allChkAvailableTS[dayNo][hour] = new JCheckBox();
                eachChkAvailableTS = allChkAvailableTS[dayNo][hour];
                eachChkAvailableTS.setHorizontalTextPosition(SwingConstants.LEFT);
                eachChkAvailableTS.setFont(new Font("Georgia", Font.PLAIN, 11));
                eachChkAvailableTS.setBackground(new Color(102, 0, 51));
                eachChkAvailableTS.setForeground(new Color(102, 255, 204));

                if (dayNo == 1 && hour == 0) {
                    eachChkAvailableTS.setText("Sun");
                } else if (dayNo == 2 && hour == 0) {
                    eachChkAvailableTS.setText("Mon");
                } else if (dayNo == 3 && hour == 0) {
                    eachChkAvailableTS.setText("Tue");
                } else if (dayNo == 4 && hour == 0) {
                    eachChkAvailableTS.setText("Wed");
                } else if (dayNo == 5 && hour == 0) {
                    eachChkAvailableTS.setText("Thu");
                } else if (dayNo == 6 && hour == 0) {
                    eachChkAvailableTS.setText("Fri");
                } else if (dayNo == 7 && hour == 0) {
                    eachChkAvailableTS.setText("Sat");
                }

                eachChkAvailableTS.setName(dayNo + ";" + hour + ";" + 1);
                eachChkAvailableTS.addActionListener(this);
                eachChkAvailableTS.addItemListener(this);
                pnlAvailableTS.add(eachChkAvailableTS);

            }
        }
        return pnlAvailableTS;
    }

    private JPanel inputAreaUnavailableTS() {
        JPanel pnlUnavailableTS = new JPanel();
        pnlUnavailableTS.setPreferredSize(new Dimension(670, 280));
        pnlUnavailableTS.setBorder(BorderFactory.createTitledBorder(BorderFactory.↩
createEtchedBorder(this.backgroundColor, this.foreGroundColor), "Select unavailable ↩
```

```
timeslots (Recurrs weekly)", TitledBorder.DEFAULT_JUSTIFICATION, TitledBorder.TOP, new ↩
Font("Georgia", Font.PLAIN, 14), new Color(102, 255, 204)));
        pnlUnavailableTS.add(new Canvas());
        pnlUnavailableTS.setBackground(new Color(102, 0, 51));
        allChkUnavailableTS = new JCheckBox[8][24];

        for (int dayNo = 1; dayNo < 8; dayNo++) {
            for (int hour = 0; hour < 24; hour++) {
                allChkUnavailableTS[dayNo][hour] = new JCheckBox();
                eachChkUnavailableTS = allChkUnavailableTS[dayNo][hour];
                eachChkUnavailableTS.setHorizontalTextPosition(SwingConstants.LEFT);
                eachChkUnavailableTS.setFont(new Font("Georgia", Font.PLAIN, 11));
                eachChkUnavailableTS.setBackground(new Color(102, 0, 51));
                eachChkUnavailableTS.setForeground(new Color(102, 255, 204));
                if (dayNo == 1 && hour == 0) {
                    eachChkUnavailableTS.setText("Sun");
                } else if (dayNo == 2 && hour == 0) {
                    eachChkUnavailableTS.setText("Mon");
                } else if (dayNo == 3 && hour == 0) {
                    eachChkUnavailableTS.setText("Tue");
                } else if (dayNo == 4 && hour == 0) {
                    eachChkUnavailableTS.setText("Wed");
                } else if (dayNo == 5 && hour == 0) {
                    eachChkUnavailableTS.setText("Thu");
                } else if (dayNo == 6 && hour == 0) {
                    eachChkUnavailableTS.setText("Fri");
                } else if (dayNo == 7 && hour == 0) {
                    eachChkUnavailableTS.setText("Sat");
                }

                eachChkUnavailableTS.setName(dayNo + ";" + hour + ";" + 2);
                eachChkUnavailableTS.addActionListener(this);
                eachChkUnavailableTS.addItemListener(this);
                pnlUnavailableTS.add(eachChkUnavailableTS);
            }
        }
        return pnlUnavailableTS;
    }

    @Override
    public void actionPerformed(ActionEvent e) {

```

UpdateEventFrame.java

```
210             Object btnSource = (Object) e.getSource();
211             if (btnSource.equals(btnRegister)) {
212                 //Null validation
                    if (txtFirstName.getText().equals(null) || txtFirstName.getText().equals("")) {
214                     JOptionPane.showMessageDialog(null, "Please enter first name.");
215                     txtFirstName.setText("");
216                     txtFirstName.requestFocus();
217                     return;
218                 }
                    if (txtLastName.getText().equals(null) || txtLastName.getText().equals("")) {
220                     JOptionPane.showMessageDialog(null, "Last Name cannot be left blank!");
221                     txtLastName.setText("");
222                     txtLastName.requestFocus();
223                     return;
224                 }
                    if (txtLocation.getText().equals(null) || txtLocation.getText().equals("")) {
226                     JOptionPane.showMessageDialog(null, "Location cannot be left blank!");
227                     txtLocation.setText("");
228                     txtLocation.requestFocus();
229                     return;
230                 }
                    if (txtEmail.getText().equals(null) || txtEmail.getText().equals("")) {
232                     JOptionPane.showMessageDialog(null, "Email cannot be left blank!");
233                     txtEmail.setText("");
234                     txtEmail.requestFocus();
235                     return;
236                 }
                    if (new String(txtPassword.getPassword()).equals(null) || new String(txtPassword.getPassword()).equals("")) {
238                     JOptionPane.showMessageDialog(null, "Password cannot be left blank!");
239                     txtPassword.setText("");
240                     txtPassword.requestFocus();
241                     return;
242                 }
243
244                 String firstName = txtFirstName.getText();
245                 String lastName = txtLastName.getText();
246                 String email = txtEmail.getText();
247                 String location = txtLocation.getText();
```

```java
248                 String password = new String(txtPassword.getPassword());
249                 ArrayList<TimeSlot> allAvailableTS, allUnavailableTS;
250
251                 //Get timeslot objects parsed from string
252                 allAvailableTS = timeslotController.parseTS(allAvailableTS_Str);
253                 allUnavailableTS = timeslotController.parseTS(allUnavailableTS_Str);
254
255                 //Send all parameters to userController to register the user.
256                 userController.registerUser(firstName, lastName, email, location, password,↩
     allAvailableTS, allUnavailableTS);
257                 JOptionPane.showMessageDialog(null, "Registration Succesfull!", "User ↩
    Regiserted!", JOptionPane.INFORMATION_MESSAGE);
258             } else if (btnSource.equals(btnReset)) {
259                 resetFields();
260
261                 for (int dayNo = 1; dayNo < 8; dayNo++) {
262                     for (int hour = 0; hour < 24; hour++) {
263                         eachChkUnavailableTS = allChkUnavailableTS[dayNo][hour];
264                         eachChkUnavailableTS.setSelected(false);
265                     }
266                 }
267                 for (int dayNo = 1; dayNo < 8; dayNo++) {
268                     for (int hour = 0; hour < 24; hour++) {
269                         eachChkAvailableTS = allChkAvailableTS[dayNo][hour];
270                         eachChkAvailableTS.setSelected(false);
271                     }
272                 }
273                 txtFirstName.requestFocus();
274             } else if (btnSource.equals(btnLogin)) {
275                 dispose();
276             }
277         }
278
279
280         //Selection validation
281         //Logic to toggle available and unavailable checkbox
282         @Override
            public void itemStateChanged(ItemEvent e) {
284
285             JCheckBox selectedCheckBox = (JCheckBox) e.getItem();
286
287             if (e.getStateChange() == ItemEvent.SELECTED) {
```

```
288                 String tokens[] = selectedCheckBox.getName().split(";");
289                 int day = Integer.parseInt((String) tokens[0]);
290                 int hour = Integer.parseInt((String) tokens[1]);
291                 int timeSlotType = Integer.parseInt((String) tokens[2]);
292
293                 if (timeSlotType == 1) {
294                     allAvailableTS_Str.add(selectedCheckBox.getName());
295
296                     for (int dayNo = 1; dayNo < 8; dayNo++) {
297                         for (int hours = 0; hours < 24; hours++) {
298                             eachChkUnavailableTS = allChkUnavailableTS[dayNo][hours];
299                             String tokens_u[] = eachChkUnavailableTS.getName().split(";");
300                             int day_u = Integer.parseInt((String) tokens_u[0]);
301                             int hour_u = Integer.parseInt((String) tokens_u[1]);
302                             if (day_u == day && hour_u == hour) {
303                                 eachChkUnavailableTS.setEnabled(false);
304                                 break;
305                             }
306
307                         }
308                     }
309
310                 } else if (timeSlotType == 2) {
311                     allUnavailableTS_Str.add(selectedCheckBox.getName());
312
313                     for (int dayNo = 1; dayNo < 8; dayNo++) {
314                         for (int hours = 0; hours < 24; hours++) {
315                             eachChkAvailableTS = allChkAvailableTS[dayNo][hours];
316                             String tokens_u[] = eachChkAvailableTS.getName().split(";");
317                             int day_u = Integer.parseInt((String) tokens_u[0]);
318                             int hour_u = Integer.parseInt((String) tokens_u[1]);
319                             if (day_u == day && hour_u == hour) {
320                                 eachChkAvailableTS.setEnabled(false);
321                                 break;
322                             }
323                         }
324                     }
325                 }
326             }
327             if (e.getStateChange() == ItemEvent.DESELECTED) {
328
329                 String tokens[] = selectedCheckBox.getName().split(";");
330                 int day = Integer.parseInt((String) tokens[0]);
```

```
331                 int hour = Integer.parseInt((String) tokens[1]);
332                 int timeSlotType = Integer.parseInt((String) tokens[2]);
333
334                 if (timeSlotType == 1) {
335                     allAvailableTS_Str.remove(selectedCheckBox.getName());
336
337                     for (int dayNo = 1; dayNo < 8; dayNo++) {
338                         for (int hours = 0; hours < 24; hours++) {
339                             eachChkUnavailableTS = allChkUnavailableTS[dayNo][hours];
340                             String tokens_u[] = eachChkUnavailableTS.getName().split(";");
341                             int day_u = Integer.parseInt((String) tokens_u[0]);
342                             int hour_u = Integer.parseInt((String) tokens_u[1]);
343                             if (day_u == day && hour_u == hour) {
344                                 eachChkUnavailableTS.setEnabled(true);
345                                 break;
346                             }
347                         }
348                     }
349
350                 } else if (timeSlotType == 2) {
351                     allUnavailableTS_Str.remove(selectedCheckBox.getName());
352
353                     for (int dayNo = 1; dayNo < 8; dayNo++) {
354                         for (int hours = 0; hours < 24; hours++) {
355                             eachChkAvailableTS = allChkAvailableTS[dayNo][hours];
356                             String tokens_u[] = eachChkAvailableTS.getName().split(";");
357                             int day_u = Integer.parseInt((String) tokens_u[0]);
358                             int hour_u = Integer.parseInt((String) tokens_u[1]);
359                             if (day_u == day && hour_u == hour) {
360                                 eachChkAvailableTS.setEnabled(true);
361                                 break;
362                             }
363                         }
364                     }
365                 }
366             }
367         }
368
369         public void resetFields() {
370             //Clear all fields
371             txtFirstName.setText("");
372             txtLastName.setText("");
373             txtLocation.setText("");
```

```java
        txtPassword.setText("");
        txtEmail.setText("");

    }

    //Class to draw hour labels on JFrame
    private class Canvas extends JPanel {

        private Canvas() {
            setPreferredSize(new Dimension(650, 40));

        }

        @Override
        protected void paintComponent(Graphics g) {
            super.paintComponents(g);
            Graphics2D g2d = (Graphics2D) g;
            g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_↩
ANTIALIAS_ON);
            g2d.setFont(new Font("Georgia", Font.ITALIC, 11));
            g2d.setColor(new Color(102, 255, 204));
            int x = 20, y = 35;

            g2d.drawString("Hourly blocks", x + 260, y - 20);
            g2d.drawString("Hourly blocks", x + 260, y + 80);

            g2d.setFont(new Font("Georgia", Font.BOLD, 14));
            for (int hr = 0; hr < 24; hr++) {
                //Draw timeslot hour labels
                g2d.drawString(hr + "", x += 13, y);
                g2d.drawString(hr + "", x += 13, y + 100);
                //Draw day labels
                int y2 = 100;
                g2d.drawString(hr + "", x, y2 += 10);
                g2d.drawString(hr + "", x, y2 += 10);
            }
        }

    }
}
```

UpdateProfileFrame.java

```java
package view;

import controller.SocialTime;
import controller.UserController;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JOptionPane;

public class UpdateProfileFrame extends JFrame implements ActionListener {

    private JPanel contentPane;
    private JTextField txtEmail;
    private JTextField txtPassword;
    private JTextField txtConfirmPassword;

    private final JButton btnSubmit;
    private UserController userController;

    /**
     * Create the frame.
     */
    public UpdateProfileFrame() {
        setTitle("Update Profile");
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setBounds(100, 100, 352, 171);
        contentPane = new JPanel();
        contentPane.setBackground(new Color(102, 0, 51));
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JLabel lblNewLabel = new JLabel("Login Email");
        lblNewLabel.setForeground(new Color(102, 255, 204));
        lblNewLabel.setBackground(new Color(102, 255, 204));
        lblNewLabel.setBounds(67, 23, 89, 14);
```

```
44          contentPane.add(lblNewLabel);
45
46          JLabel lblPassword = new JLabel("New Password");
47          lblPassword.setForeground(new Color(102, 255, 204));
48          lblPassword.setBackground(new Color(102, 255, 204));
49          lblPassword.setBounds(49, 48, 107, 14);
50          contentPane.add(lblPassword);
51
52          JLabel lblConfirmNewPassword = new JLabel("Confirm new Password");
53          lblConfirmNewPassword.setForeground(new Color(102, 255, 204));
54          lblConfirmNewPassword.setBackground(new Color(102, 255, 204));
55          lblConfirmNewPassword.setBounds(10, 72, 146, 14);
56          contentPane.add(lblConfirmNewPassword);
57
58          txtEmail = new JTextField();
59          txtEmail.setBounds(182, 18, 125, 20);
            contentPane.add(txtEmail);
61          txtEmail.setColumns(10);
62
63          txtPassword = new JTextField();
64          txtPassword.setColumns(10);
65          txtPassword.setBounds(182, 43, 125, 20);
66          contentPane.add(txtPassword);
67
68          txtConfirmPassword = new JTextField();
69          txtConfirmPassword.setColumns(10);
70          txtConfirmPassword.setBounds(182, 67, 125, 20);
71          contentPane.add(txtConfirmPassword);
72
73          btnSubmit = new JButton("Submit");
74          btnSubmit.setBounds(218, 98, 89, 23);
            btnSubmit.addActionListener(this);
76          contentPane.add(btnSubmit);
77
78          userController = SocialTime.getUserController();
79
            setLocationRelativeTo(null);
            setVisible(true);
82
83      }
84
```

```
85          @Override
            public void actionPerformed(ActionEvent e) {
87              Object btnSource = e.getSource();
88              if (btnSource.equals(btnSubmit)) {
89                  //Null validation
                    if (txtEmail.getText().equals(null) || txtEmail.getText().equals("")) {
91                      JOptionPane.showMessageDialog(null, "Please enter email.");
92                      txtEmail.setText("");
93                      txtEmail.requestFocus();
94                      return;
95                  }
                    if (txtPassword.getText().equals(null) || txtPassword.getText().equals(""))↩
    {
97                      JOptionPane.showMessageDialog(null, "Password cannot be set blank!");
98                      txtPassword.setText("");
99                      txtPassword.requestFocus();
100                     return;
101                 }
102                 if (!txtPassword.getText().equals(txtConfirmPassword.getText())) {
103                     JOptionPane.showMessageDialog(null, "Passwords did not match!");
104                     txtPassword.setText("");
105                     txtConfirmPassword.setText("");
106                     txtPassword.requestFocus();
107                     return;
108                 }
109                 String newEmail = txtEmail.getText();
                    String newPassword = new String(txtPassword.getText());
111
112                 UserController.getLoggedInUser().setEmail(newEmail);
113                 UserController.getLoggedInUser().setPassword(newPassword);
114                 userController.updateProfile(UserController.getLoggedInUser());
115                 JOptionPane.showMessageDialog(null, "Please use newly set email & password ↩
    from next login onwards.", "Profile updated!", JOptionPane.INFORMATION_MESSAGE);
116                 this.dispose();
117             }
118         }
119     }
120
```

Conclusion

This documentation has presented the analysis, design, and implementation for – “Social Time”, an event planner system developed in Object-oriented software development paradigm. Static analysis was done using class diagrams at the domain and architecture level. NLA was done to infer domain level classes. Use case was used to identify functions of system at high level. Activity diagram showed diagrammatically the algorithm for generating feasible timeslots. And, finally, the design was translated into Java in the implementation phase.

Referencing

- Schach, S. (2011). Object-oriented and classical software engineering. 1st ed. Boston [u.a.]: McGraw-Hill.
- Bruegge, B. and Dutoit, A. (2014). Object-oriented software engineering. 1st ed. Harlow: Pearson.
- Bell, D. & Parr, M., 2010. Exceptions. In: *Java for Students, 6th Ed..* London: Pearson Education Limited, pp. 301-314.
- Deitel, P. & Deitel, H., 2015. Regular Expressions, Class Patterns and Class Matcher. In: *Java How To Program, 10th Ed..* New Jersey: Pearson, pp. 624-633.
- Schildt, H., 2014. Painting in Swing. In: *Java The Complete Reference, 9th Edition.* New York: McGraw-Hill Education, pp. 1036-1040.
- Vermeulen, A. et al., 2000. Documentation Conventions. In: *The Element of Java Style.* Cambridge: Cambridge University Press, pp. 31-52.