Table of Contents

[Table of figures 1](#_Toc480900587)

[Abstract 2](#_Toc480900588)

[Introduction 2](#_Toc480900589)

[Task 1 – List of potential/candidate classes with final class diagram 3](#_Toc480900590)

[NLA 3](#_Toc480900591)

[Step 1 - Noun identification for potential classes 3](#_Toc480900592)

[Step 2 – Verb identification for potential methods/operations 9](#_Toc480900593)

[Step 3 – Grouping methods and attributes to domain classes 13](#_Toc480900594)

[Step 4 – Drafting class diagram 14](#_Toc480900595)

[Task 2 – Activity Diagram of feasible timeslot 16](#_Toc480900596)

[Task 3 – Use Case 18](#_Toc480900597)

[Task 4 – Code Architecture 19](#_Toc480900598)

[Task 5 – System implementation: Java source code 20](#_Toc480900599)

[Package – controller 21](#_Toc480900600)

[Package – dao 30](#_Toc480900601)

[Package – dbutil 48](#_Toc480900602)

[Package – model 49](#_Toc480900603)

[Package – model.factory 60](#_Toc480900604)

[Package – subject\_observer 65](#_Toc480900605)

[Package – view 66](#_Toc480900606)

[Conclusion 104](#_Toc480900607)

[Referencing 105](#_Toc480900608)

# Table of figures

[Figure 1:Class Diagram Legend 15](#_Toc480899693)

[Figure 2: Class Diagram (Domain Level) 16](#_Toc480899694)

[Figure 3: Activity Diagram for feasible timeslot generation 18](#_Toc480899695)

[Figure 4: Use Case - Social Time 19](#_Toc480899696)

[Figure 5: Class Diagram (Architecture Level) 20](#_Toc480899697)

# 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.

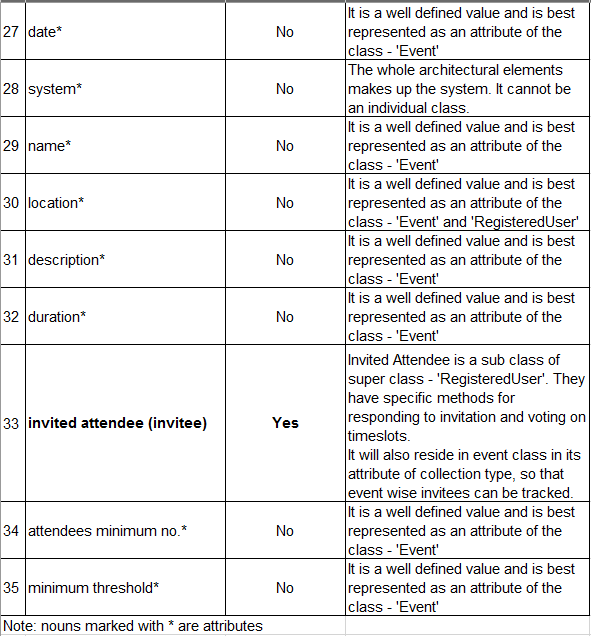
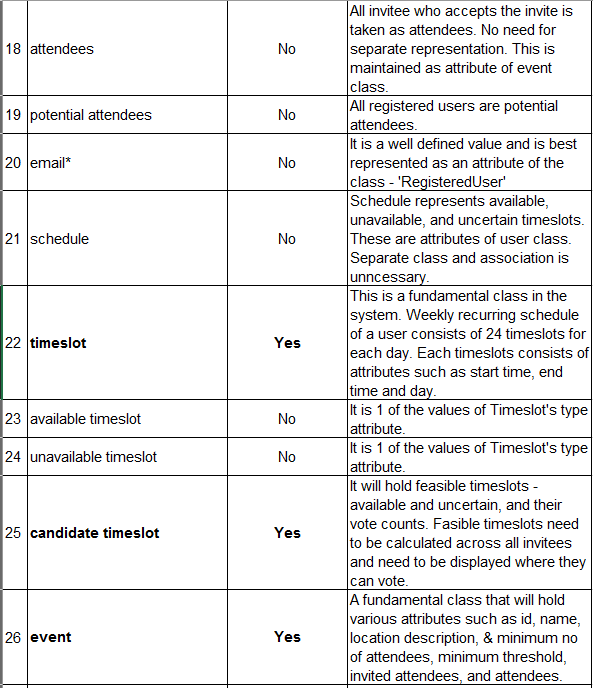
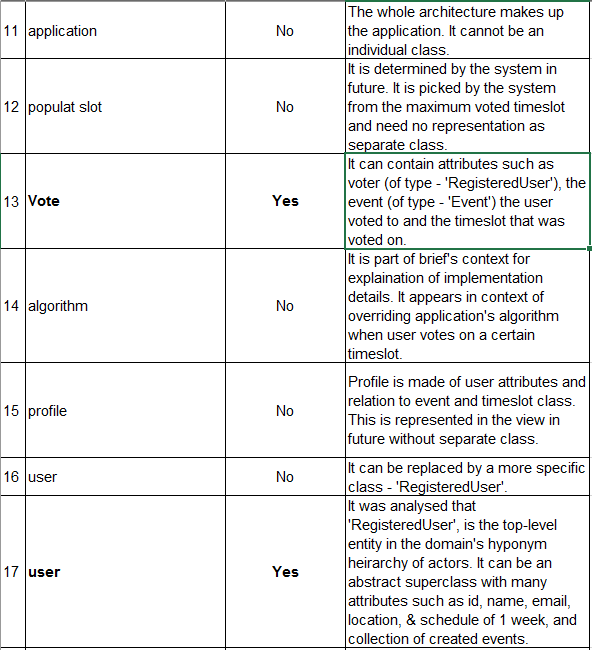
# Task 1 – List of potential/candidate classes with final class diagram

## 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.

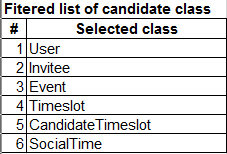
## Step 1 - Noun identification for potential classes

* 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.)



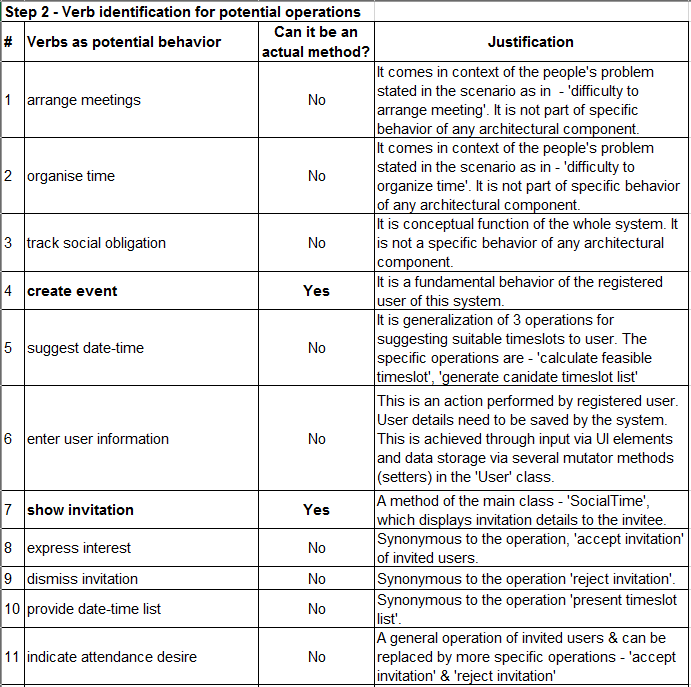
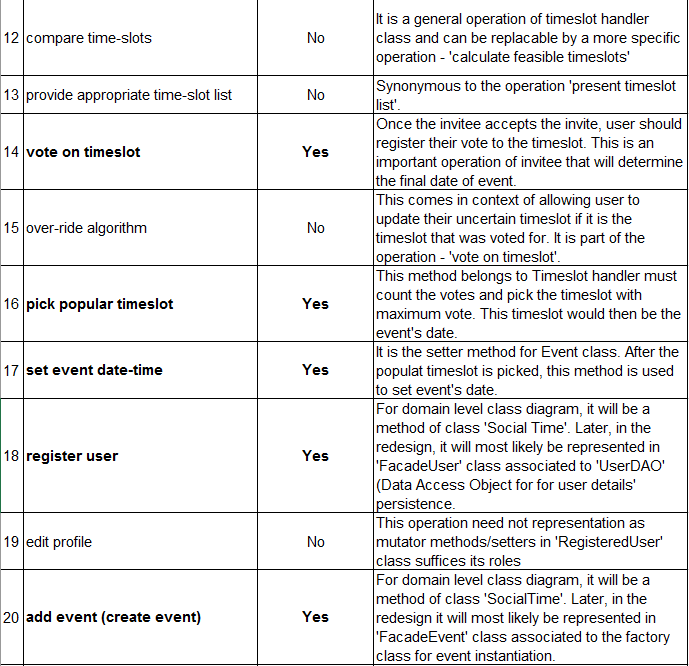
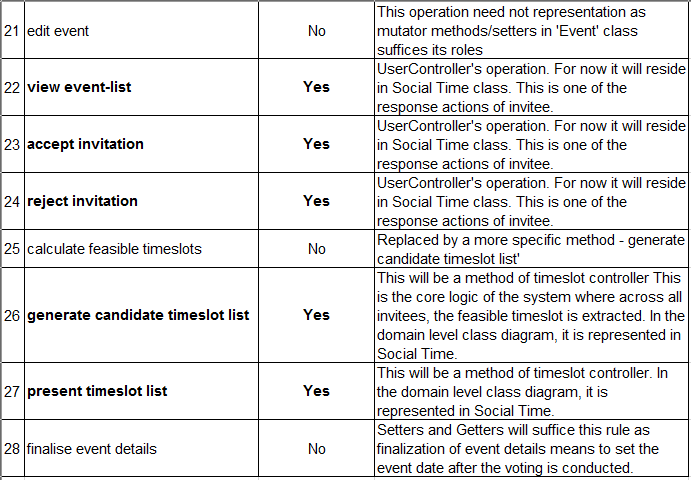
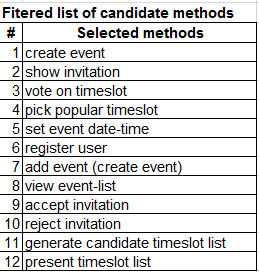
**Adjectives**

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

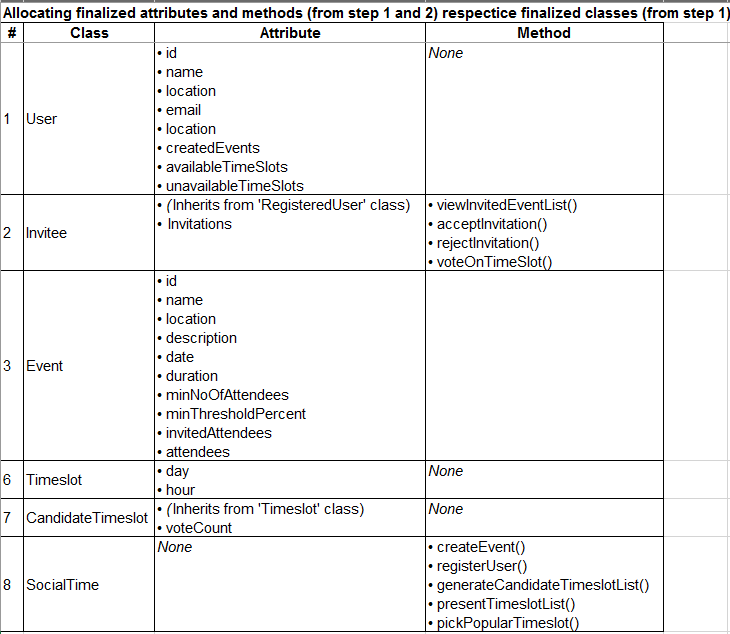


## Step 2 – Verb identification for potential methods/operations

* 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 3 – Grouping methods and attributes to domain classes



## 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. |
| C:\Users\Biju Ale\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Realization.png | Depicts interface implementation. |
| Uni-directional association | Depicts unidirectional association between classes. |
| C:\Users\Biju Ale\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Bi-directional association.png | Depicts bidirectional association between classes. |
| C:\Users\Biju Ale\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Aggregation.png | Depicts 1 class aggregates another without binding its existence in own life. |
| C:\Users\Biju Ale\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Composition.png | Depicts 1 class composes another binding its existence in its own life. |
| C:\Users\Biju Ale\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Dependency.png | Depicts 1 class depends on another in such way that the change propagates to dependent. |

Figure 1:Class Diagram Legend

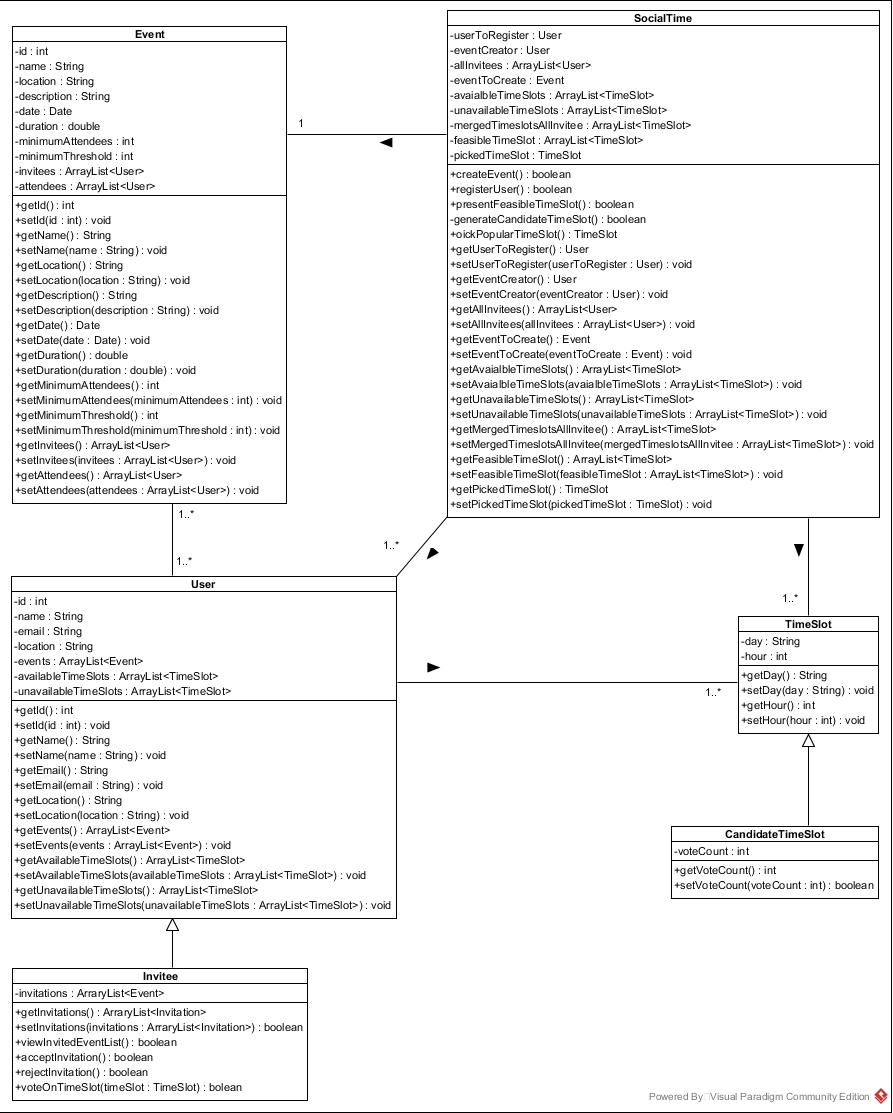


Figure 2: Class Diagram (Domain Level)

# Task 2 – Activity Diagram of feasible timeslot

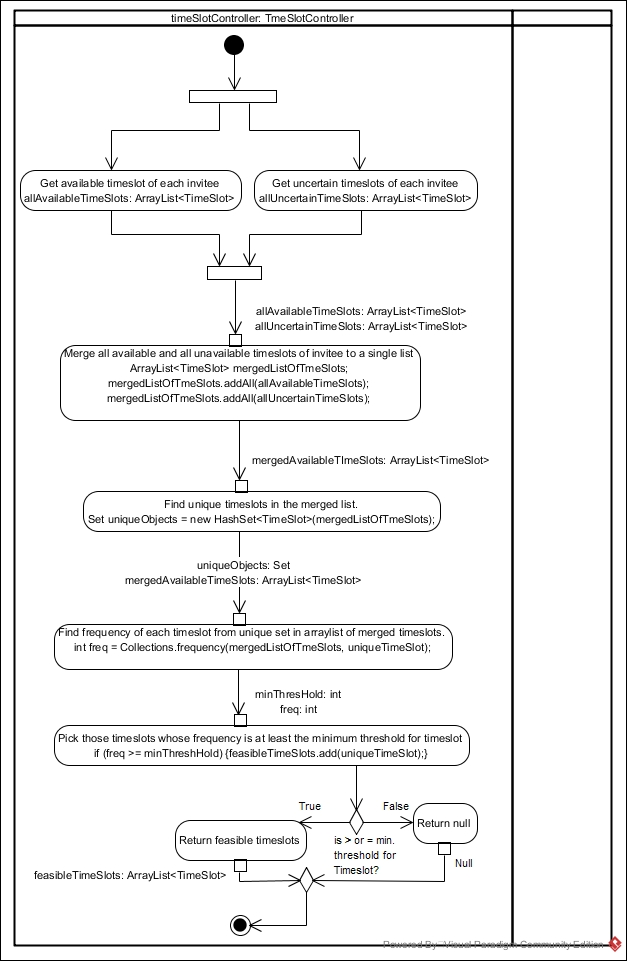


Figure 3: Activity Diagram for feasible timeslot generation

# 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.

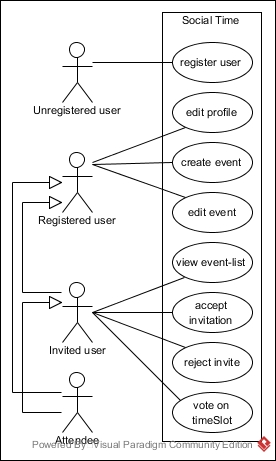
****

Figure 4: Use Case - Social Time

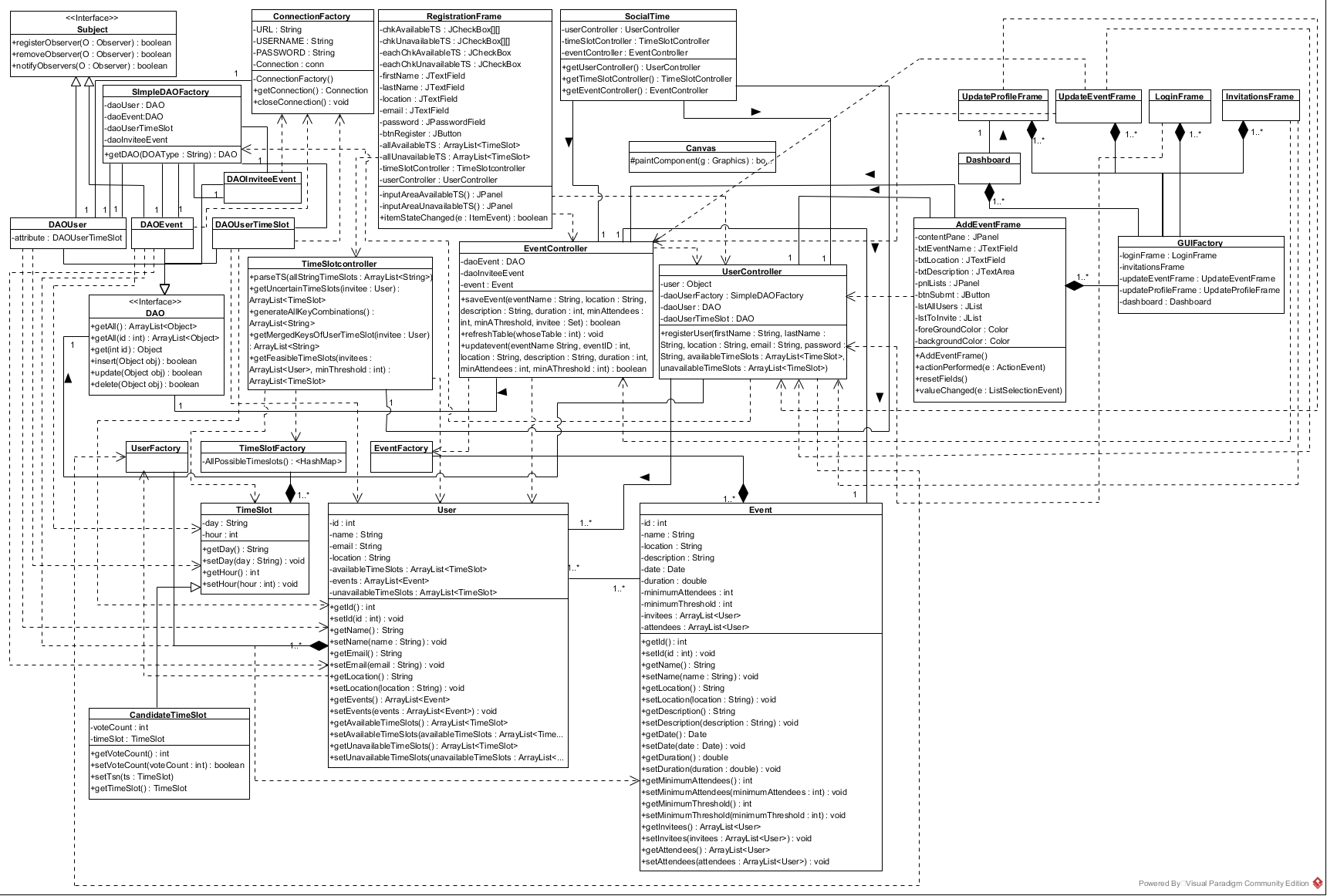
Task 4 – Code Architecture  


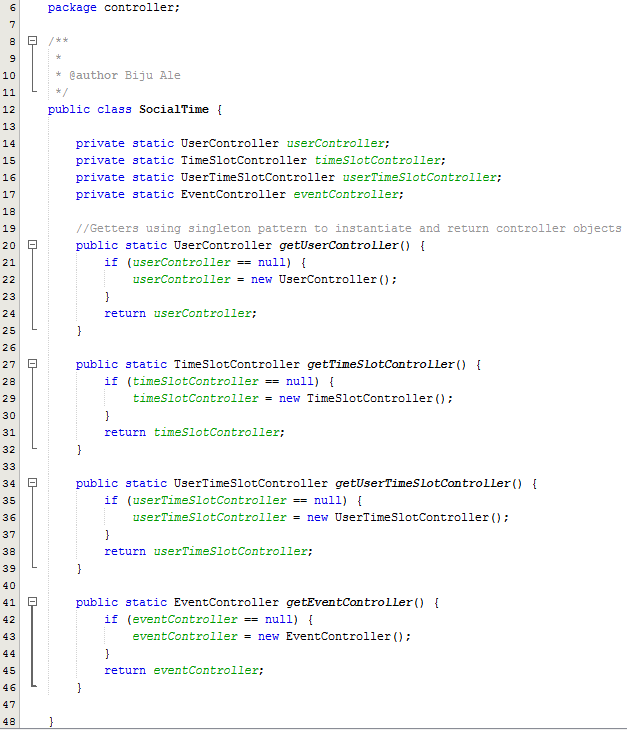
Figure 5: Class Diagram (Architecture Level)

# Task 5 – System implementation: Java source code

### Package – controller

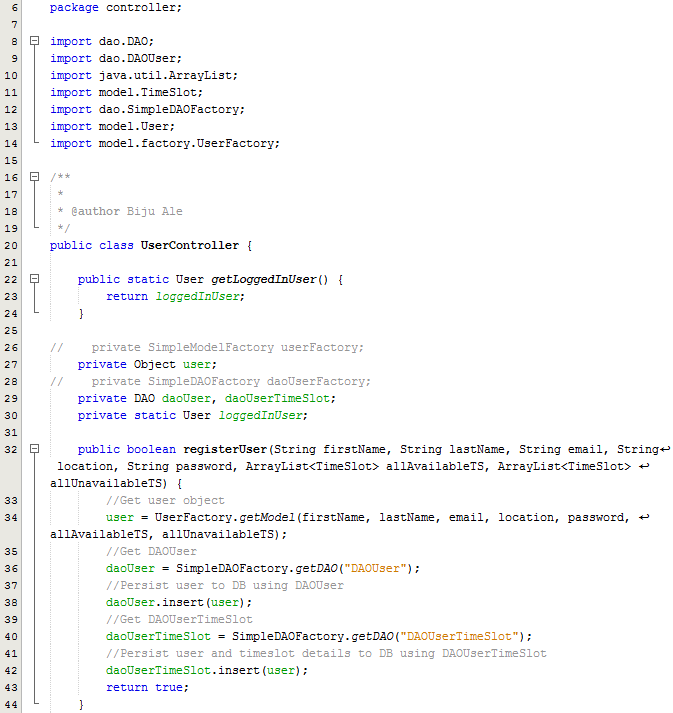
#### EventController.java

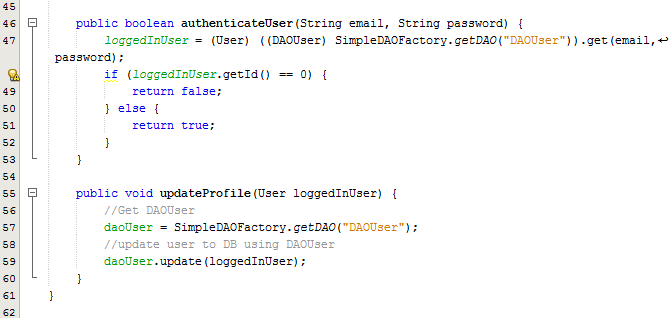
#### SocialTime.java



#### TimeSlotController.java

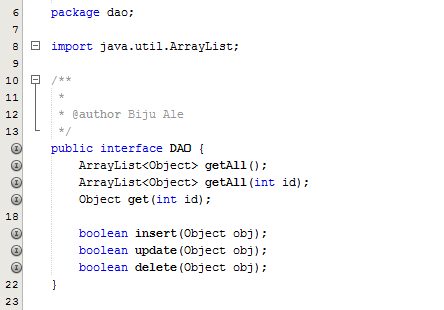
#### UserController.java



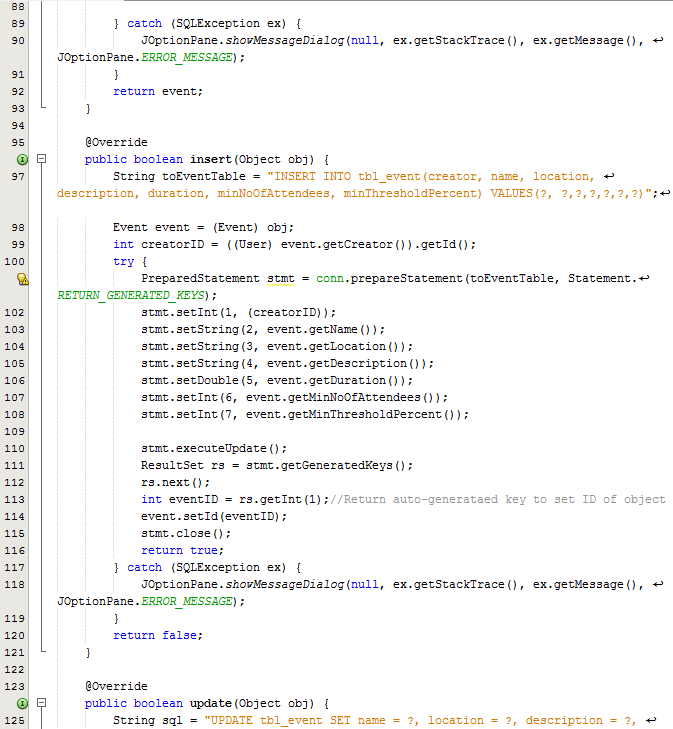


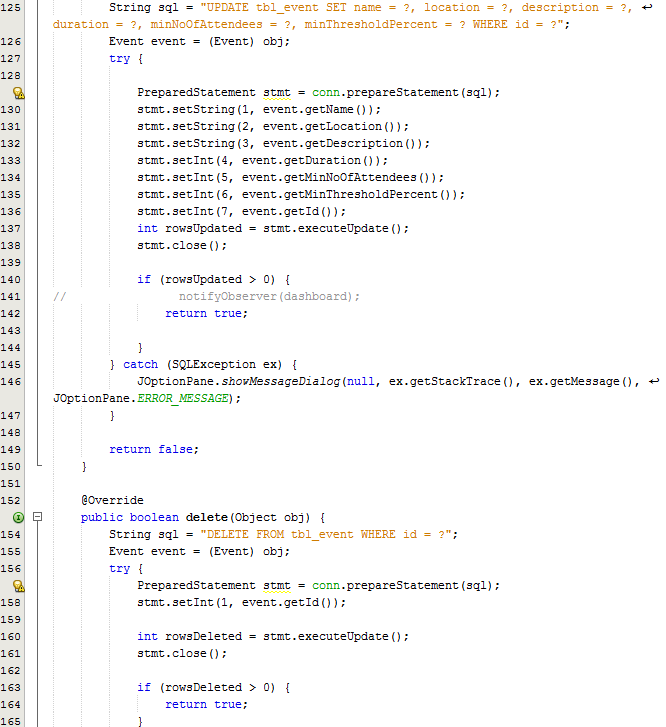
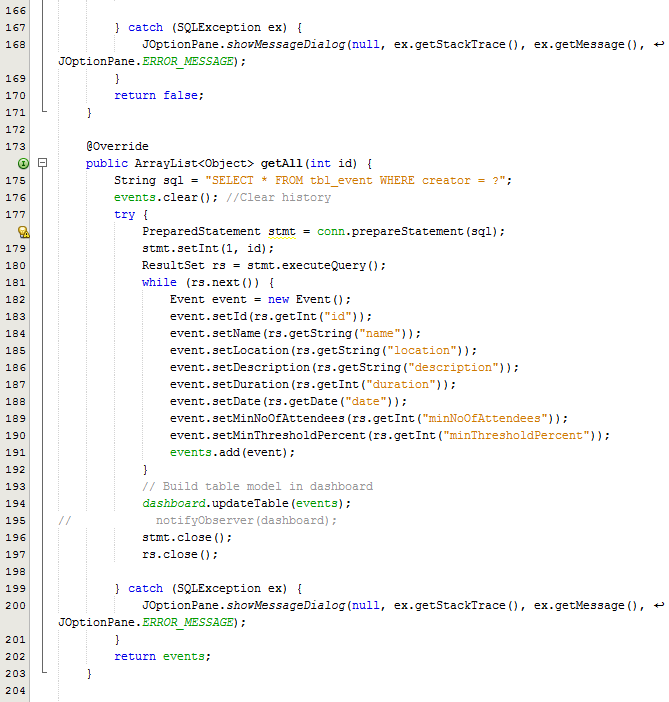
### Package – dao

#### DAO.java (Interface)



#### DAOEvent.java



#### DAOInviteeEvent.java

#### DAOUser.java

#### DAOUserTimeSlot.java

#### SimpleDAOFactory.java



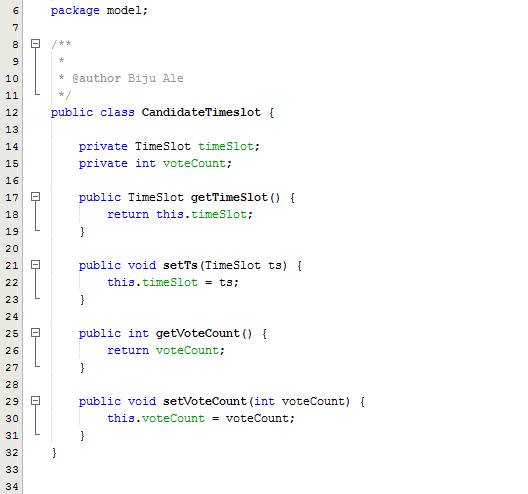
### Package – dbutil

#### ConnectionFactory.java

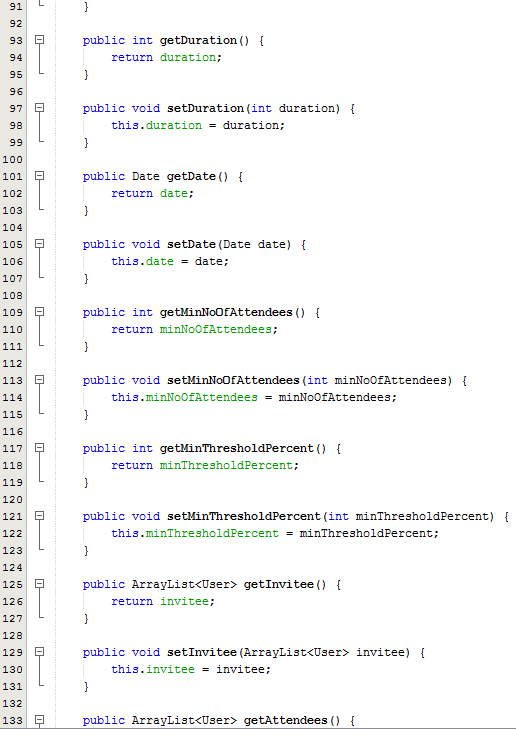


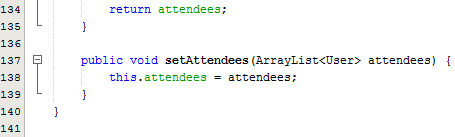
### Package – model

#### CandidateTimeslot.java

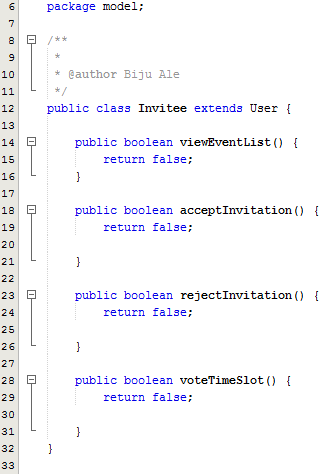


#### Event.java





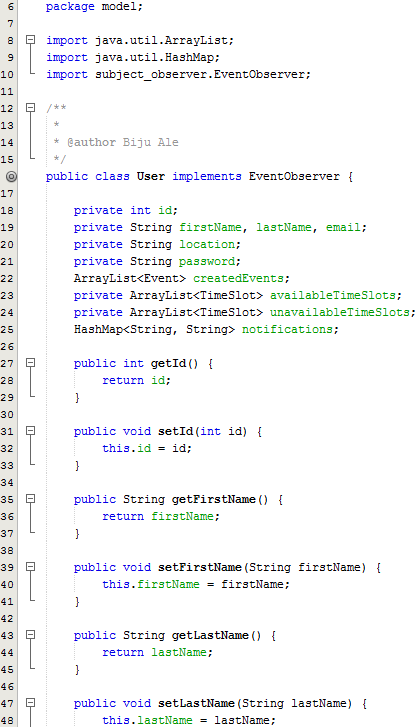
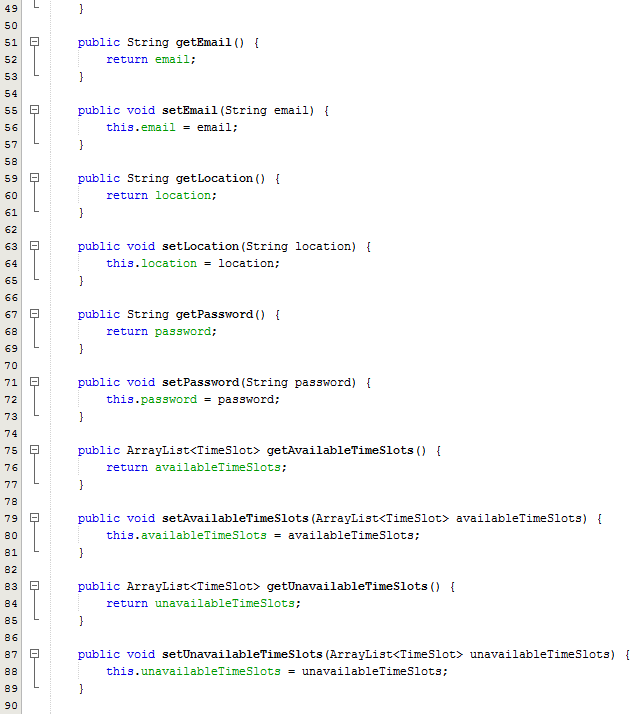
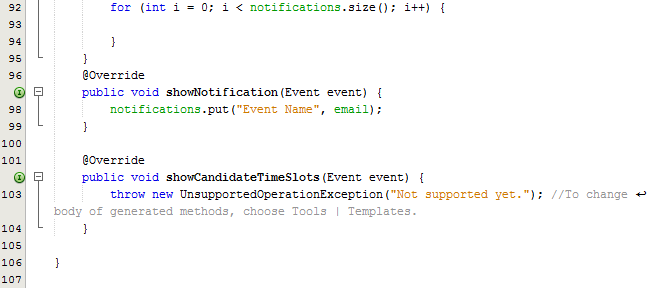
#### Invitee.java

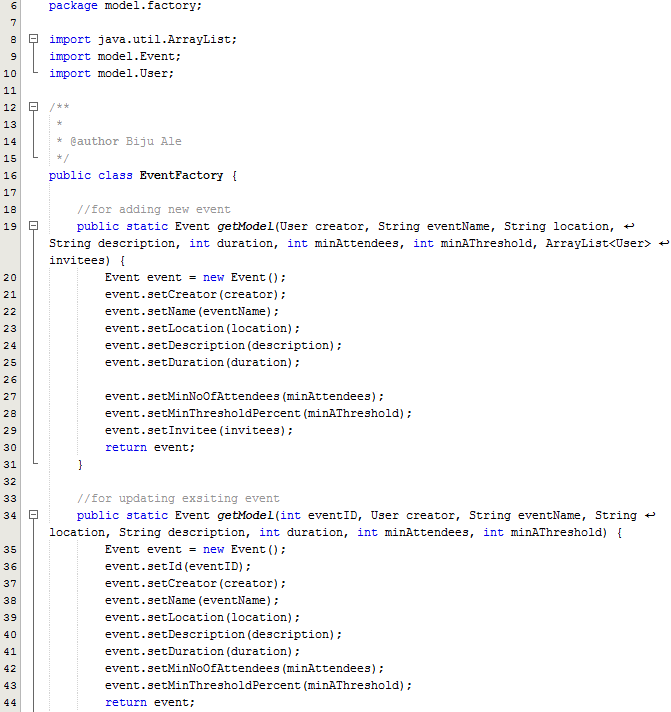


#### TimeSlot.java



#### User.java

Package – model.factory  
  
 

#### EventFactory.java



#### TimeSlotFactory.java

#### UserFactory.java

### Package – subject\_observer

#### EventObserver.java

#### Observer.java (interface)

#### Subject.java (interface)

### Package – view

#### AddEventFrame.java

#### Dashboard.java

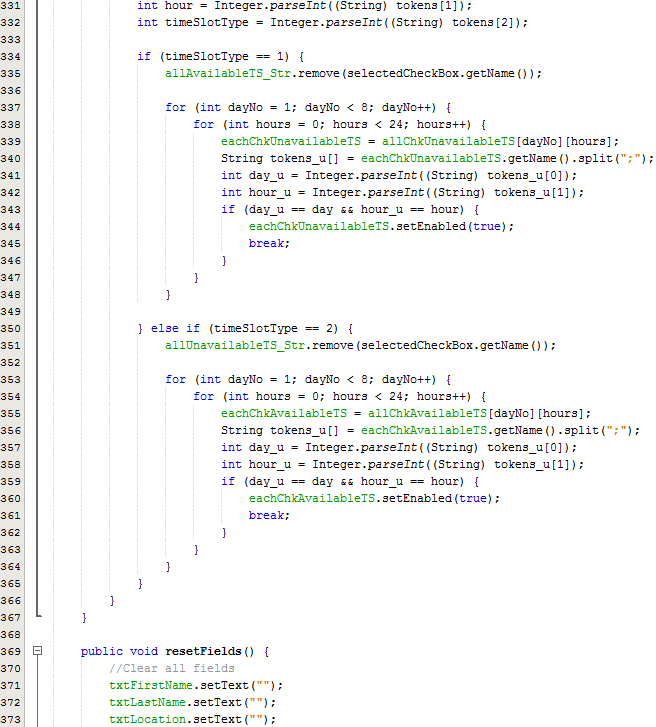
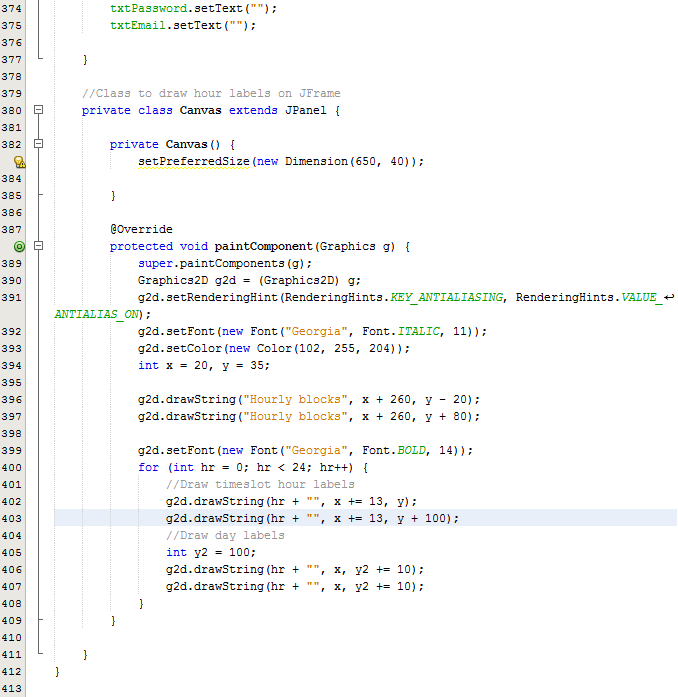
#### GUIFactory.java

#### InvitationsFrame.java

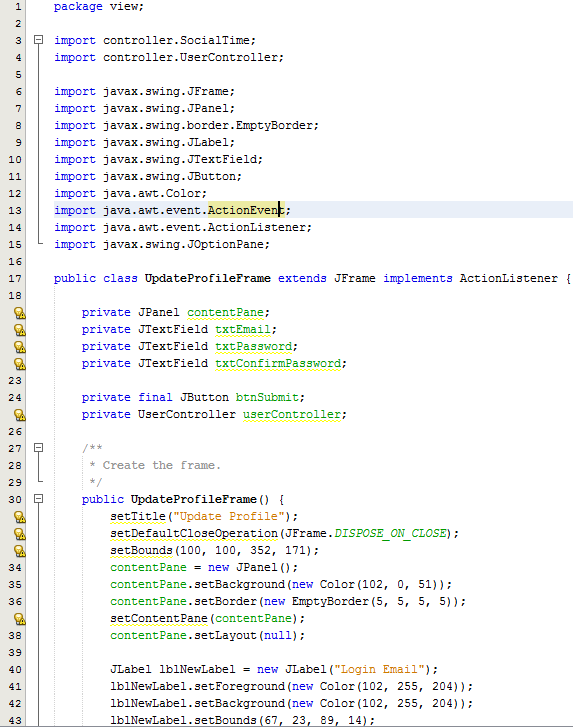
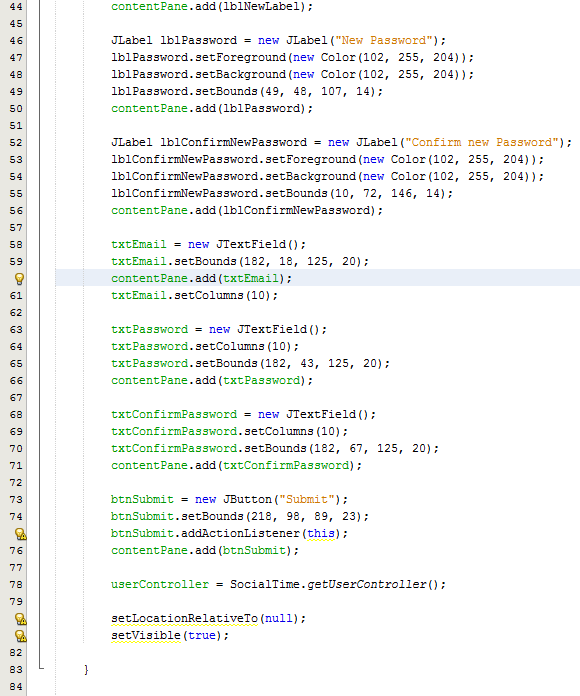
#### LoginFrame.java

#### RegistrationFrame.java

#### UpdateEventFrame.java

#### UpdateProfileFrame.java

# 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.