System design document for Calendar++

Version: 1.0

Date: 2015-05-31

Authors: Johan Ben Mohammad, Erik Forsberg, Patrick Franz, Michael Tran

This version overrides all previous versions.

Contents

1 Introduction

This is the system design document (SDD) for Group 14's project "Calendar++".
The document will describe how the application works and how it is built.

1.1 Design goals

Simplicity and ease of maintenance are key goals in the development process.

Some key principles of programming that we have tried to follow are:

- Single responsibility principle (SRP): A class should only be responsible for a single part of the program.

- Interface-segregation principle (ISP): The model consists of many relatively small interfaces classes. This means that the system is easier to understand, maintain and possibly extend.
- Model View Controller (MVC): The program is split into three conceptual parts. The model, which is the representation of the program itself. The view, which is the user interface implementation. The view can request and display data from the model. The controller, which sends commands to the model in order to manipulate its state.
- Data access object (DAO): The program/domain communicates with a DAO layer which provides specific data operations that manipulates the database. This prevents the domain/program from directly communicating with the database which can expose details of the database.
- Unit testing with mock objects (using JUnit)

The model should be tested to some degree in order to make it less likely to generate faults and errors.

1.2 Definitions, acronyms and abbreviations

- GUI: Graphic user interface
- DAO: Data access object
- JPA: Java persistence API

2 System design

2.1 Overview

The application is built with the MVC (model view controller) design pattern in mind. This pattern separates the user interface from the model and controller code. With this technique, the user interface can be replaced more easily to suit other platforms such as mobile phones and/or tablets and still use the same model code.

The “top” object is the CalendarPlus model class. This object contains references to the other classes in the model.

The model uses interfaces to clearly state what functionality an object of its kind must implement. A example of this is the mock objects we use for testing. These objects are hard coded and we have full control over them, but they still implement the same contract as the actual classes we use. This means that they can safely be used for testing.

A note on testing: The classes in the model have been tested with a few test each. Our model classes are very simple and they consists mostly of set and get methods. When we introduced the database, the tests will modify this database and cause various errors. This means that some tests will not pass, even if they should pass, and have passed in the past. Therefore we have decided to comment out some tests to run the program more easily.

2.1.2 Managers

To avoid circular dependencies, “Managers” have been created to associate certain parts of the model without making them depend on each other. These are:

- ActivityManager: HashMap with an activity as key and a list of attendees (contacts) as value. This hashmap represents which contacts are listed participants in an activity.
- ContactManager: HashMap with a contact as key and a list of contact groups as value. This hashmap helps us list the contact groups a contact is part of.
- NotificationManager: HashMap with an activity as key and a notification as value: This hashmap ties a specific activity with its corresponding notification.

2.1.3 Event handling and controllers

The view classes have their corresponding controllers (except the listview). Code that modifies the view are kept in the view classes itself. When any of the view classes need to call methods in the model, the corresponding controller handles this.

2.2 Software decomposition

2.2.1 General

- CalendarPlusPlus: top level, including main class.
- View package: view classes
- Controller package: controller classes
- Mode package: model classes, including manager classes.
- Persistence package: database handling classes

See 2.2.3 for hierarchy.

2.2.2 Decomposition into subsystems

N/A. The program is decomposed into the Model, the view and the controller subpackages. We have no unified subsystems.

2.2.3 Layering

Top level

```
src.edu.chl.calendarplusplus
        main.java
        src.edu.chl.calendarplusplus.controller
                [controller classes]
        src.edu.chl.calendarplusplus.model
                [model classes]
        src.edu.chl.calendarplusplus.view
                [view classes]
        src.edu.chl.calendarplusplus.persistance
                [database classes]

test.edu.chl.calendarplusplus
        test.edu.chl.calendarplusplus.model
                [model test classes]
        mockclasses
                [model mock classes]
```

2.2.4 Dependency analysis

Dependencies are shown as in Figures below. There are some circular dependencies between the model package and the persistence package. This is because we use the data access object (DAO) pattern with JPA and EclipseLink library, which requires the persistence package to use the model package and vice versa. There are however no circular dependencies between the MVC-pattern.

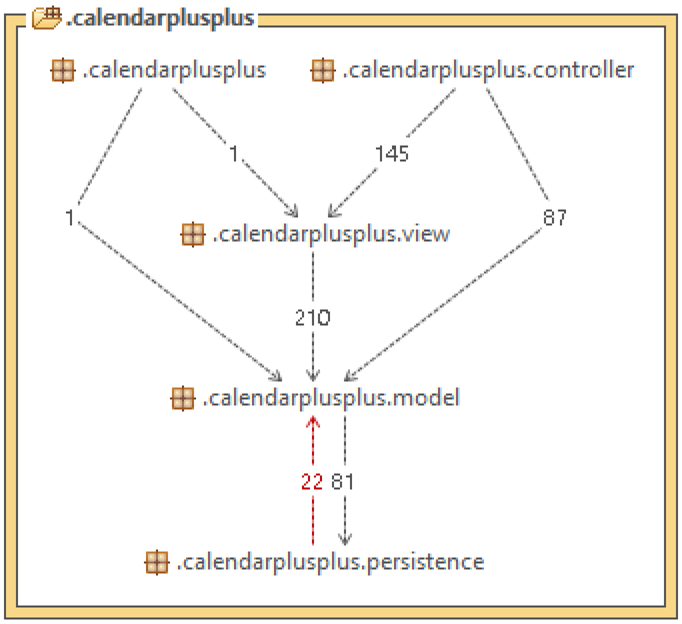

Figure 1: Member Dependency Analysis using STAN

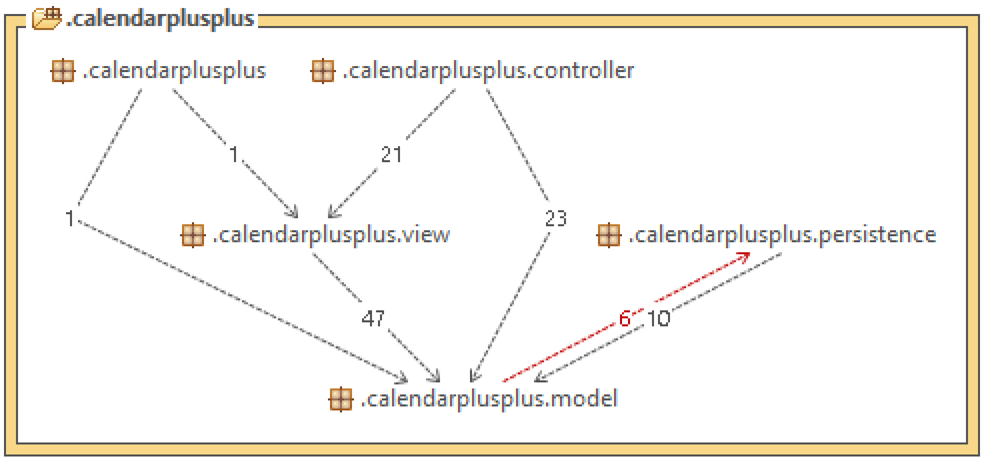

Figure 2: Class Dependency Analysis using STAN

2.3 Concurrency issues

N/A. Single threaded application.

We had a goal to create a thread that handled the alarms and notifications. This thread would sleep during a certain period (until an alarm or notification occurred) and then show a message dialog of some sort to notify the user. Unfortunately, we did not have time to implement this feature.

2.4 Persistent data management

All persistent data will be stored, retrieved and manipulated in a local derby database that is created (a folder named calendarDB) once the program is run, and located in the same folder as the application. The program uses Embedded Derby JDBC Driver to access the database within the application, and libraries such as JPA together with EclipseLink to manipulate it.

2.5 Access control and security

N/A. This a short term school project and will not focus on security.

2.6 Boundary conditions

N/A. The program is launched as a standard desktop application.

3 References

MVC: http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

DAO: http://en.wikipedia.org/wiki/Data_access_object

JPA: http://en.wikipedia.org/wiki/Java_Persistence_API

ISP: http://en.wikipedia.org/wiki/Interface_segregation_principle

SRP: http://en.wikipedia.org/wiki/Single_responsibility_principle

EclipseLink: http://en.wikipedia.org/wiki/EclipseLink

APPENDIX

ER-Diagram:

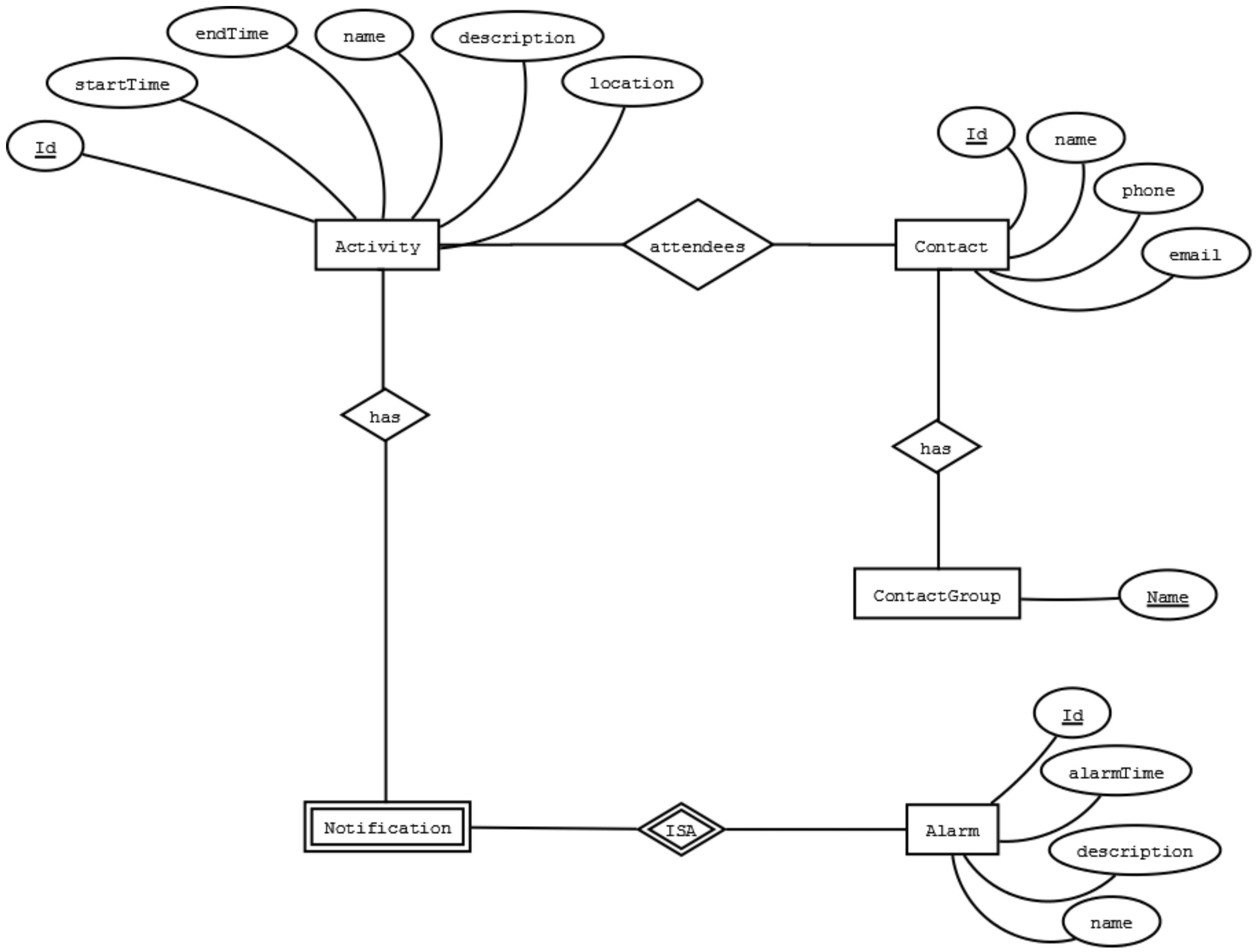

Figure 3: ER-Diagram