QuickManage
Software Architecture Document

Version 2.2

Revision History

Date	Version	Description	Author(s)
19/03/2013	2.0	First published version	Trinh Luan, Ngo Tan Thinh, Nguyen Manh Hung, Cao Phi Hung, Dao Tien Minh
14/04/2013	2.1	Second published version with more details and clarifications based on the first published version as well as new features added to the program	Trinh Luan, Ngo Tan Thinh, Nguyen Manh Hung, Cao Phi Hung, Dao Tien Minh
01/05/2013	2.2	Third published version aims to provide more information and updates in the logical view of the software. Classes diagrams are added to better explain the design of the program.	Trinh Luan, Ngo Tan Thinh, Nguyen Manh Hung, Cao Phi Hung, Dao Tien Minh

Table of Contents

Software Architecture Document

1. Introduction

This is a technical document about the QuickManage program. This document intends to cover and explain significant decisions that has been made on the system by examining different architectural views. The views include use case view, logical view, and data view. Besides, the document also describe the drivers of the architectural decisions such as functional, non-functional requirements and system's constraints.

1.1 Scope

The scope of QuickMange project is to create a usable, reliable, and user-friendly content management system (CMS) to help staffs and managers at a music school to manage their students.

The scope of this document is to examine the software from top down and further explain architectural decisions made during the development of the program. In addition, the document shall be updated frequently so that readers can use this document as a reliable reference to study the architecture of the software.

1.2 Targeted Audience

This document is for technical readers such as software engineers, programmers, and project managers and anyone who wish to learn about the architecture of the program.

Readers of this document should be familiar with UML diagrams and notations.

1.3 Architectural Representation

This document uses UML as a language to illustrate and explain the architecture of the system. The document does not provide any description on UML. For more information about UML, please visit the following link: http://www.uml.org/.

1.4 References

This document was written based on its previous version and some other resources. Below is the tables with more details information on the resources used to write the document.

Document Name	Date (dd/mm/yyy)	Author
QuickMange SAD Version 2.1	01/05/2013	NEXT
UML 2.0	22/03/2013	www.uml.org
Android Bot SAD Version 1.0	29/03/2013	Chamika Weerasinghe
Software Architecture Document	29/03/2013	Andrew McDowell

for SOLA Development Snapshot		
Model-View-Controller	01/04/2013	Microsoft http://msdn.microsoft.com/en-us/library/ff649643.aspx

2. Architectural Overview

The overall architecture of the program is based on Model-View-Controller (MVC) pattern. This pattern allows the program to be developed in different and independent parts: the models, the views, and the controllers.

The models manage the data and perform operations on data of the program. The models will respond to request from the views and instructions from the controllers. The controllers interpret users' actions and instruct the models to respond appropriately. The views help to display data from the model and fire events to the controllers.

By separating the program in to three different and independent parts, the development team can work on different parts of the program at the same time with less conflict and each member in the team can work with his/her strength such as GUI or logic design. In addition, because each part of the program is independent from each other, so the program will be more flexible and adaptable to changes in the requirement as well as migration to a different system. Last but not least, MVC pattern also helps to test the program faster, because it separates the logics which is mainly in the models of the program.

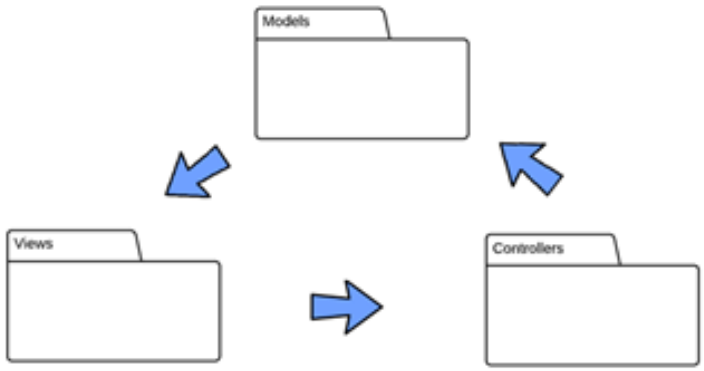


Figure1. MVC pattern used in QuickManage.

3. Architectural Drivers

This section describes the architectural drivers that have been identified for QuickMange. The drivers include architecture requirement that has architectural significance and constrains that the system must cope with. The drivers provide a rationale for key aspects of the program architecture.

3.1 Requirement

3.1.1 Functional Requirement

The table below identifies and describes the functional requirements of QuickManage that have architectural significance

NO.	Requirement	Description	Significance
1	Type of users and data	• The system must consist of objects of Mangers, Staffs, Students, Teachers, and Classes • Mangers and Staffs are the ones that directly use and interact with the system • Each Manger and Staff must have accounts with basics information and a username and a password to login and use the system • Students, Teachers, and Classes are objects to be saved, read, and modified at will by Mangers and Staffs • Mangers can perform all actions that a Staff can and Managers can also perform some actions that a Staff cannot (for detailed information about what Mangers/ staff can do, please refer to SRS document)	• Because objects like Mangers, Staffs, Students, and Teachers of the system must store basics information such as ID, name, DOB etc. Therefore, a super class called User must be created to store basic information • Staff, Student, and Teacher must inherit the User class, so the classes can have properties of their super class and extend their own properties • Manager class inherits Staff class so that it can have properties, methods from both User and Staff class, and it can extend its own properties and methods
2	Storage of data	• Managers, Staffs, Students, Teachers, and Classes have to be stored so that existing Managers, Staffs can login and use the system. Staffs, Students, and classes' information can be displayed when the program start up • Database is not allowed	• Because no database is allowed in this project, so all objects must be saved into files. • Each type of objects is stored in a different files for easy access and avoid error when loading up the system

3	Load data	• All data must be loaded into memory when the program starts to ensure fast and access to all objects created previously	• A Data class is created and a load data method is implemented in the same class to meet the requirement • The most important properties of data class is the array lists to store all the objects loaded from files. Array lists are choose over array because number of objects can change any moment while the program is running. • Load data method read all Manager, Staff, Student, Teacher and Classes files, then assign all objects in the files into array list of appropriate objects
4	Save data	• All objects either newly created or lately modified has to be saved in order for the program to be up-to-date. • Staffs and Managers can perform this action	• A save data method is implemented in Data class to save objects created or modified • The method must loop through appropriate file(s) and find the specified objects to be saved • Every time an object is created or modified, the method is used to save the object
5	Edit data	• All objects contains fields of information and must be able to be edited by the direct users of the system: Managers and Staffs • Staffs and Mangers can perform this action	• An edit data method is implemented in Data class to edit objects • Only one specific object can be edited at a time • The method is called every time an object is edited • The method will loop through appropriate file(s) to find and retrieve data from a specified object • Save method is called consequently after edit data is invoked
6	Delete data	• All objects must be able to be deleted • An object must not be able to delete itself. i.e manager1 cannot delete his/her own account	• A delete method is implemented in Data class to delete object(s) • One or many objects can be deleted at the same time • The method go through all the appropriated file(s) to look for specific object(s) and remove

			the object(s) form array list • Save method is called afterward to save newly modified array list • The view is designed such that it does not provide any mechanism for the a user to delete his/her own account
7	Student Enrollment	• Student can be enrolled into one or many classes • Students and Classes have to keep reference to each other, number of classes a student can enroll is unlimited and number of student within a class is also unlimited • There are two ways to enroll a student into a class: from each individual student and from class • Enrolling student from class is not yet 100% complete, but will be completed upon completion of the project	• Managers and Staffs can choose a specific students to enroll him/her into a class • Time table view has to be implemented to show all available classes in the system • Or Managers and Staffs can enroll many students at a same time using class view • In order to enroll students from within a class, a list of students has to be created and updated in the class view. Another separate view to allow Managers/Staffs to choose student from must also be implemented • Array list of type Class is kept inside student • Array list of type Student is kept inside class
8	Teacher Assignment	• A class can only have one teacher • A teacher can teach many classes: from each individual teacher and from within a class • There are two ways to assign a teacher into a class • Assigning teacher from class is not yet 100% complete, but will be completed upon completion of the project	• Managers and Staffs can choose a specific teacher to assign him/her into a class • Time table view has to be implemented to show all available classes in the system • Or Managers and Staffs can assign teacher using class view • In order to assign teacher from within a class, a list of teacher has to be created and updated in the class view. Another separate view to allow Managers/Staffs to choose teacher from must also be implemented • Array list of type Class is kept inside teacher • A teacher is kept inside each class
9	Invoice Generation	• Managers/Staffs can generate invoice(s) for one	• InvoiceView, InvoiceModel, InvoiceController classes are

		or many students • Invoice will be automatically saved and associated with a student • Managers/Staffs can view all invoices of a student • Invoice will be marked as paid if paid by the student • Invoice must include note to write any additional notes • Only manager can change invoice from paid to unpaid • More than one invoice can be generated for a day	created inside View, Model, Controller packages • An array list of invoice type is implemented in Student class in Model
10	Report Generation	• Managers/Staffs can generate report(s) for one or many students • Reports consist of 2 parts: students who paid and students who have not paid • Report is not associated with any student • Report can be printed by month • Only manager can print report • Report will not be saved	• ReportChooseForm, ReportForm, ReportChooseFormController, ReportFormController classes are created in both View and Controller package • ReportChooseForm established a view for user to choose a specific month for the report • ReportForm display the report before printing it • ReportChooseFormController and ReportFormController interpret user's action on the view and send signal to generate report when the user wants
11	Dash Board	• Will appear when user log in to the system • Provide a summary about the number of classes, students, teachers, current month paid tuition fee, current month unpaid tuition fee, last month paid tuition fee, last month unpaid tuition fee and utilization ratio • Utilization ratio = rooms' occupied hours/ total available rooms' hours	• DashBoardView class created to provide the view • All raw info can be obtained from Data class so calculation is done directly inside DashBoardView • No any other class is required to complete the requirement.

12	Room	• Class Room includes room numbers, type, and occupied time • A room can be either 1-1 or 1-many • Room is needed to calculate utilization ratio	• Room class is created with properties like room number, type, and occupied time • This class is associated with Classes class and ClassType class
13	Class Type	• Each class created should have a type • Class type determines fee, number of lessons a week • Managers can add/edit/delete class type	• ClassType class is created with properties: name, feePerLesson, remarks, lesson • ClassType associates with Classes class • Classes have different fees depends on ClassType
14	Teacher hourly pay rate	• Teachers might have different salary depending on the influence of him/her in specific skills	• TeacherForm class will be changed to cope with this new requirement

15	Pay split	• Both staff and manager can generate monthly pay slip for teachers • Pay slip is calculated based on classes that he/she teach in a particular month • Play slip contain necessary information about to clearly explain the total salary of the month • Only manger can delete pay slip	• PaySlipVjew is created for each teacher, so users can view them • All raw info can be obtained from Data class so calculation is done directly inside PaySlipView • No any other class is required to complete the requirement.
16	English and Vietnamese Interface	• Users(Staff/Manger) can switch between English/Vietnamese interface	• LanguageController class is created to handle event fire from Language menu items

3.1.2 Non-functional requirements

The table below identifies and describes the non-functional requirements of QuickManage that have architectural significance.

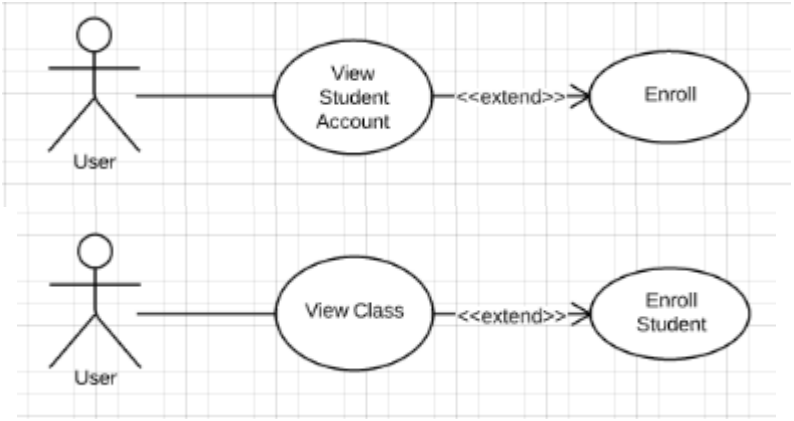
NO.	Requirement	Description	Significance
1	Persistence storage	• Data must be saved automatically after each operation on the data is done to avoid data losses • Data must be saved as binary files	• Each type of objects such as Manager, Staff, Student, Teacher, Class is saved under a binary files • The separation in types of objects ensure data can be loaded easily and avoid errors while loading
2	Safe handling	• The program should be able to detect invalid inputs and respond appropriately • The program should not stop working because of invalid inputs or any other reasons	• A class called DataValidation is created to validate all input fields of a form • There are five main methods in the class: validateManagerForm(), validateStaffForm(), validateStudentForm(), validateTeacherForm(), validateClassForm().

			• Each method above takes in appropriate form object to validate the existed fields • Some additional methods also added to validate whether the inputs are useful or not. i.e. checkClassTime(startTime, endTime) method is to evaluate input time to see if start time and end time of a class is reasonable or not • The class methods are called, when a user is creating or editing a form, to validate input fields
3	GUI	• The GUI must be easy to use with standard menu bar and tool bar • Resolution must be 1024x768 to fit the client's screen	• Some ideas taken from Microsoft outlook to simplify the view and bring new experience for users • All views have been carefully designed to fit the resolution on client's screen
4	MVC and O-O	• Program uses MVC pattern and O-O concepts to the achieve the best results	• The whole program was designed with MVC in mind as mentioned previously • O-O concepts used widely, yet selectively.
5	Contact Person	• Each student must have a contact person	• Information about a contact person is stored inside Student model as properties
6	Unit test	• Unit tests are required to test classes in Model package	• Test classes are created inside Test package to test appropriate classes in model • All tests for model classes have passed successfully

4. Use Case View

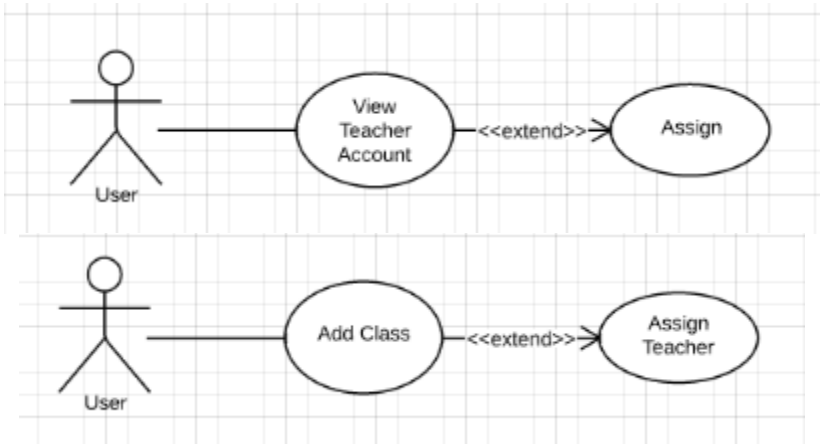
This section identifies those use cases that represent significant, and central functionality of the system. The use case also illustrate and address important points of the architecture. The following is the architectural significant use cases.

4.1 UC026 Enroll the Student



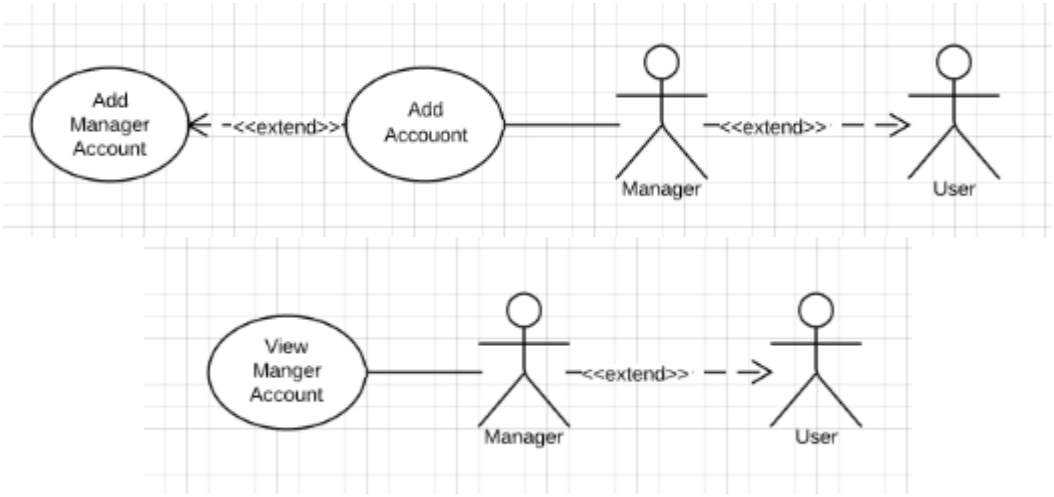
Managers/Staffs can enroll student into a class from both Student and Class view. Because user can enroll student from both view. The views and their models have to be constantly updated to any change. That's why array list of type Class in student and array list of type student in class are necessary to for quick and easy access.

4.2 UC025 Assign Teacher to Class



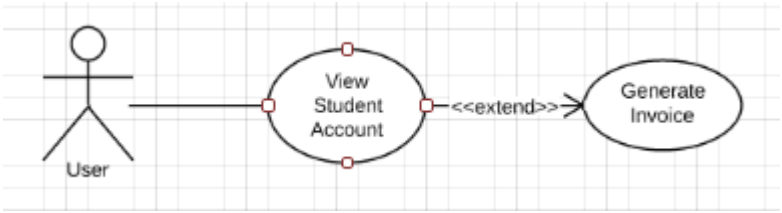
Managers/Staffs can also assign teacher into a class from both Teacher and Class view. Because user can assign teacher from both views. The views and their models have to be constantly updated to any change. That's why array list of type Class in teacher and reference to a teacher in class are necessary to for quick and easy access.

4.3 UC001 Add Manager Account and UC006 View Manger Account:



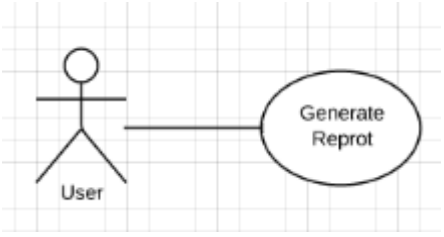
Only Managers can view and add Managers accounts. However, can perform all other function that a Staff can because Manager inherits from User (which is Staff) in this case. Inheritance makes the codes and objects highly reusable in the program.

4.4 UC032 Generate Invoice



Managers and Staffs can generate invocie for a particular student. New classes are introduced into the system to support invoice generattion. InvoiceView, InvoiceModel, InvoiceController are created in Model, View, Controller packages. Model, View, and Controller of Invoice class work closely together.

4.5 Generate Report



Managers and Staffs can generate report for a selected month. New Calsses are also inroduced into the system to support reprot generation. ReportForm, ReportChooseForm, ReportFormController, ReportChooseFormController classes are introduced. No model is needed for the report because invoices are loaded and stored inside an array list in Data class.

5. Logical View

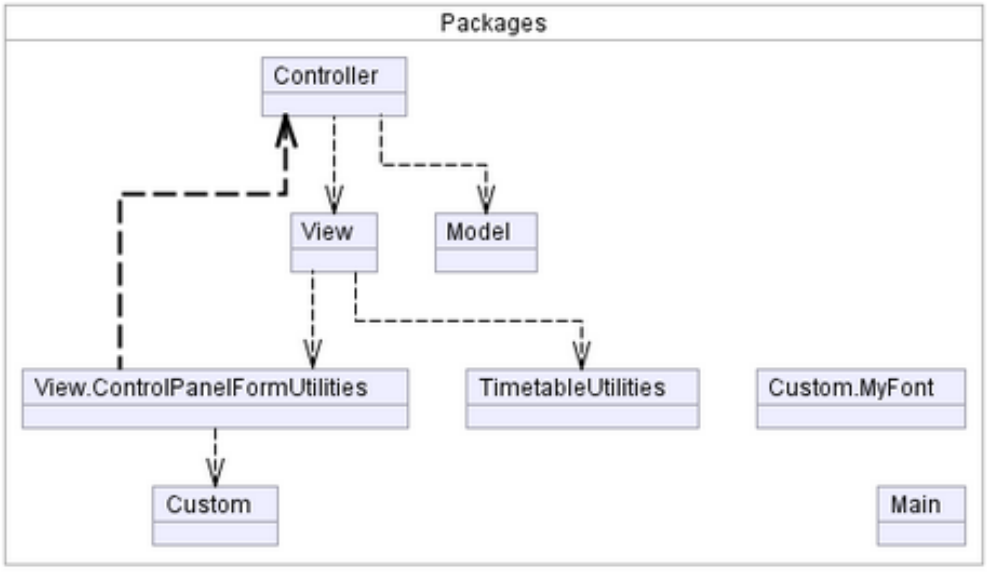
This section describes the architecturally significant parts of the design and its decomposition into packages and subsystems. For some significant packages the logical view is further decomposed into interfaces and classes.

5.1 Overview:

The program consists of important packages:

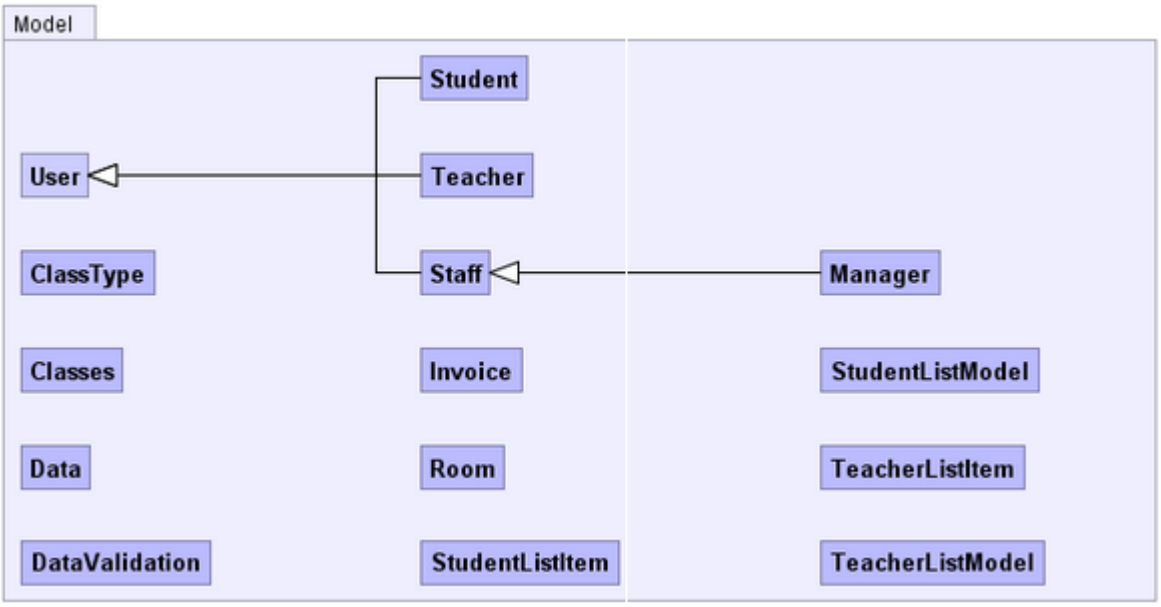
1. Controller
2. View
3. Model
4. View.ControlPanelFormUtilities
5. TimetableUtilities
6. Custom
7. Main
8. Custom
9. Test
10. Image
11. Default

The following diagram shows the high level important packages that make up the backbone of QuickManage:



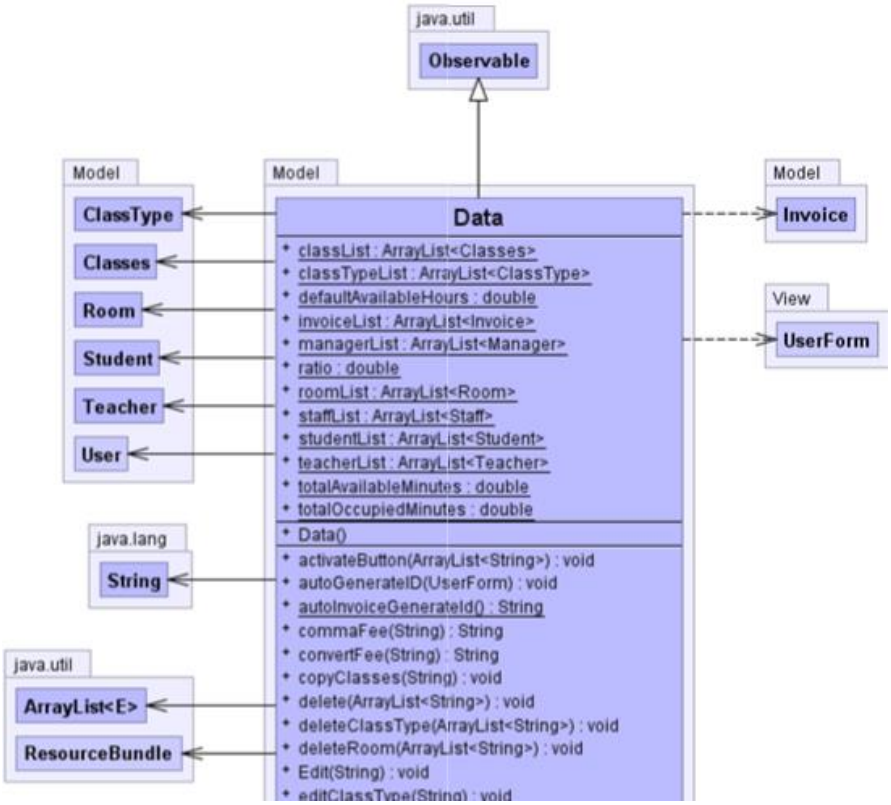
5.2 Model

Model package contains classes with properties to store data input and methods to process them. Classes in this package have close relation to each other and to other classes in View and Controller packages.



5.2.1 Data

This class is important to the program as a whole because it is responsible to store array list of Mangers, Staffs, Students, Teachers, and Classes objects. The objects are loaded into the array lists through its own loadFile() method. The class also contains method to save, edit, and delete object from the files.

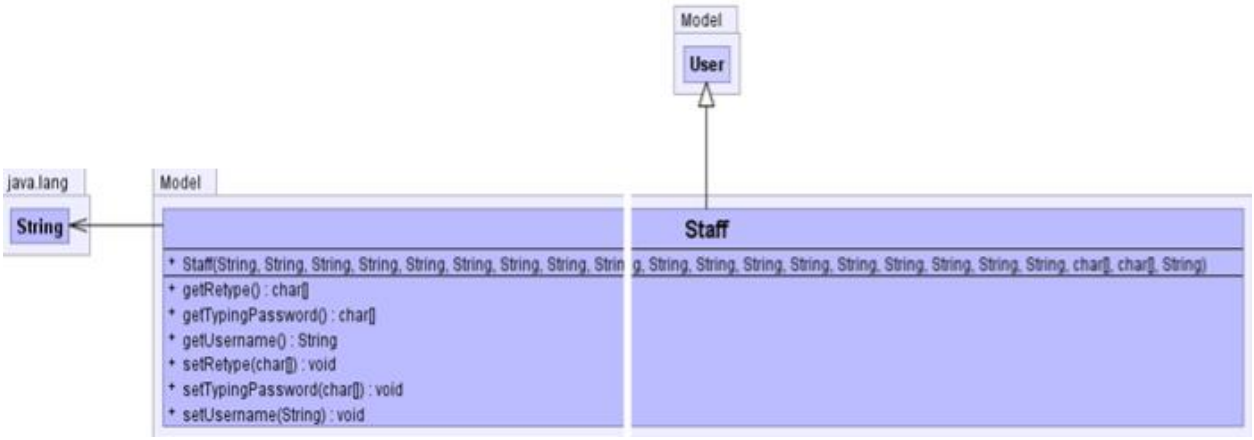


5.2.2 User

User is an abstract class that contains most of needed information for a user of the system. User class must be abstract because there its properties alone is not enough for a user in the system. Therefore, User class must be extended to create the real user.

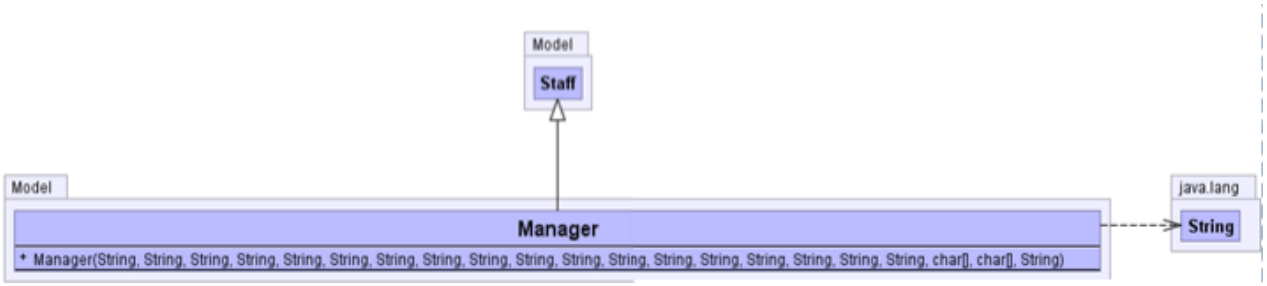
5.2.3 Staff

Staff class represents an instance of a staff. It contains all the properties inherited from User class. In addition, Staff has more properties such as user name and password to login and directly use the system.



5.2.4 Manager

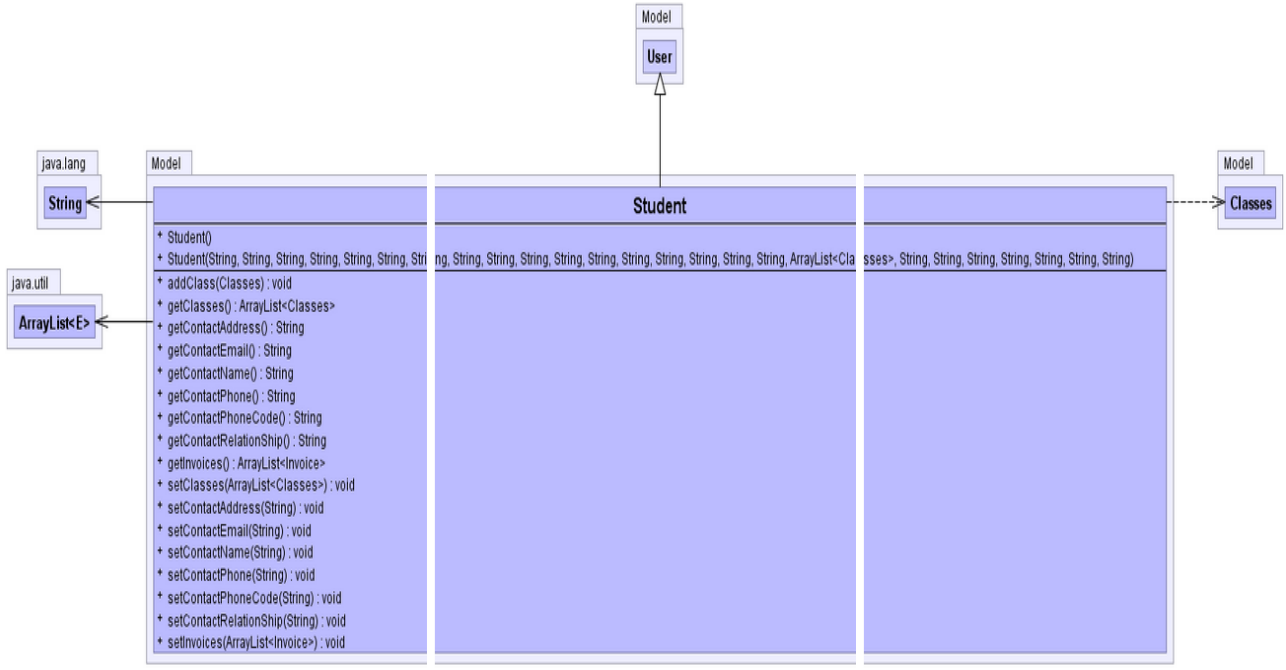
Manager class represents an instance of a manager. It contains all the properties inherited from User class. In addition, Manager has more properties such as user name and password to login and directly use the system.



5.2.5 Student

Student class represents an instance of a student. It contains all the properties inherited from User class. In addition, Student has more properties.

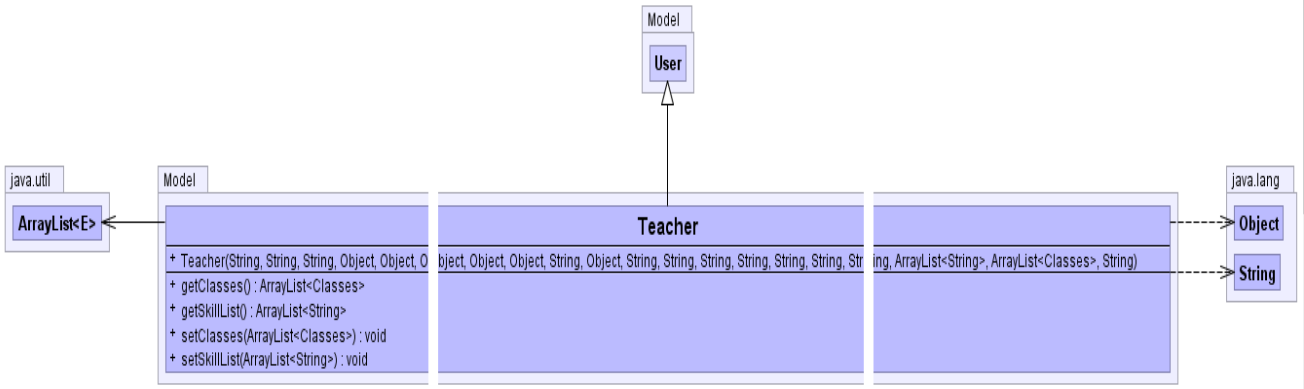
Student class has an array list of type Class to keep track of the classes that student enrolled in.



5.2.6 Teacher

Teacher class represents an instance of a teacher. It contains all the properties inherited from User class. In addition, Teacher has more properties.

Teacher class also has an array list of type Class to keep track of the classes that him/her is teaching. This class contains array list as properties to store Classes, and skill for each teacher. Teacher class also uses String and Object type, especially in its constructor to initialize properties when the class is created.



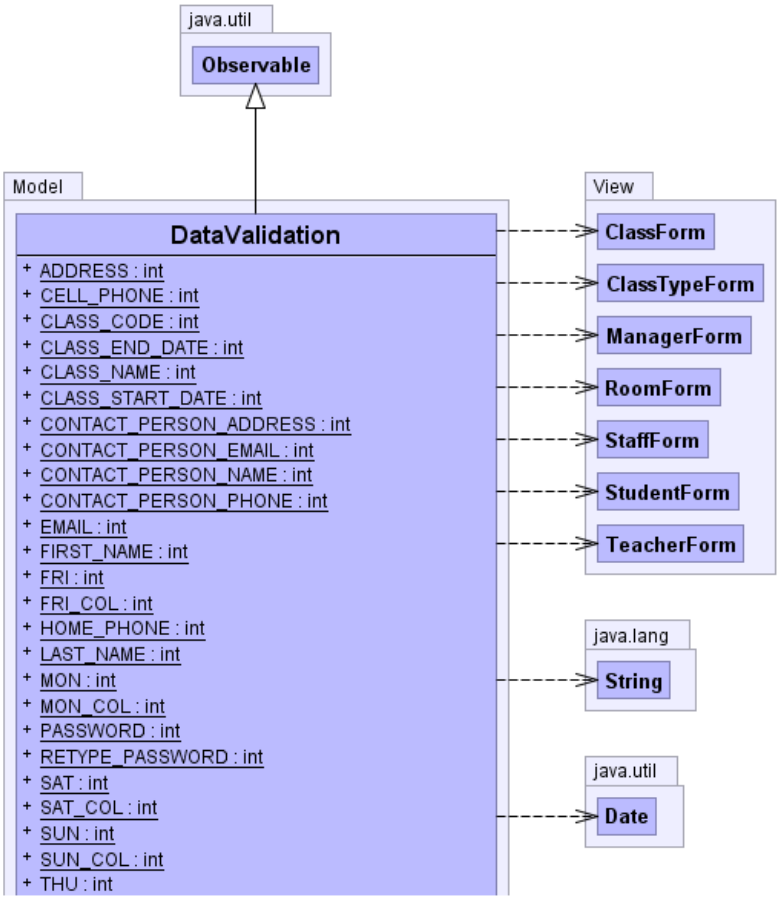
5.2.7 Classes

Classes class represents a class in the school, it contains properties that represents data of a real music class such as class name, class code, class type, start date and end date.

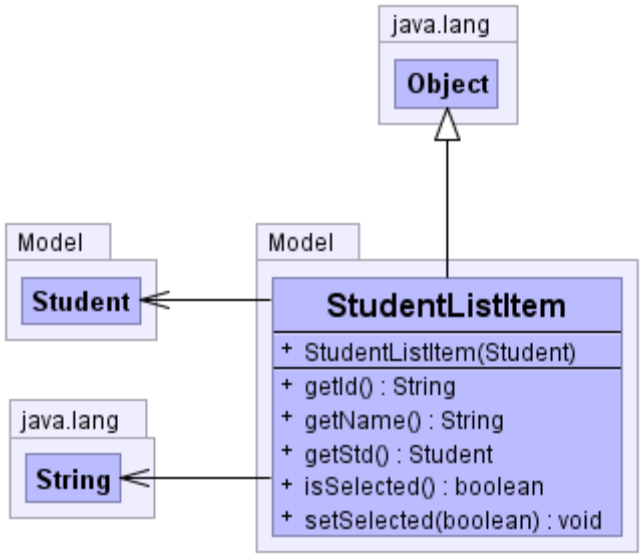
The class has an array list of students to keep track of the student currently enrolled in the class and a reference to a teacher who is teaching this class. The class uses ClassForm to get inputs from the user and implements Serializable to directly save array lists.

5.2.8 DataValidation

Another important class in Model package is DataValidation. This class contains all the method to validate input data from users such as name, phone, email, etc. The reason to have a whole class for validation is to centralize validation methods to make them more reusable, easier to access, and easier to debug. This class has its methods' parameters use many views from View package to access their properties fast and easily for validation. Datavalidation extends Observable to use setChanged() and notifyObservers() to tell the view to update accordingly based on results of the current validation. Datavalidation uses Data class to call on Data static variables, and methods.

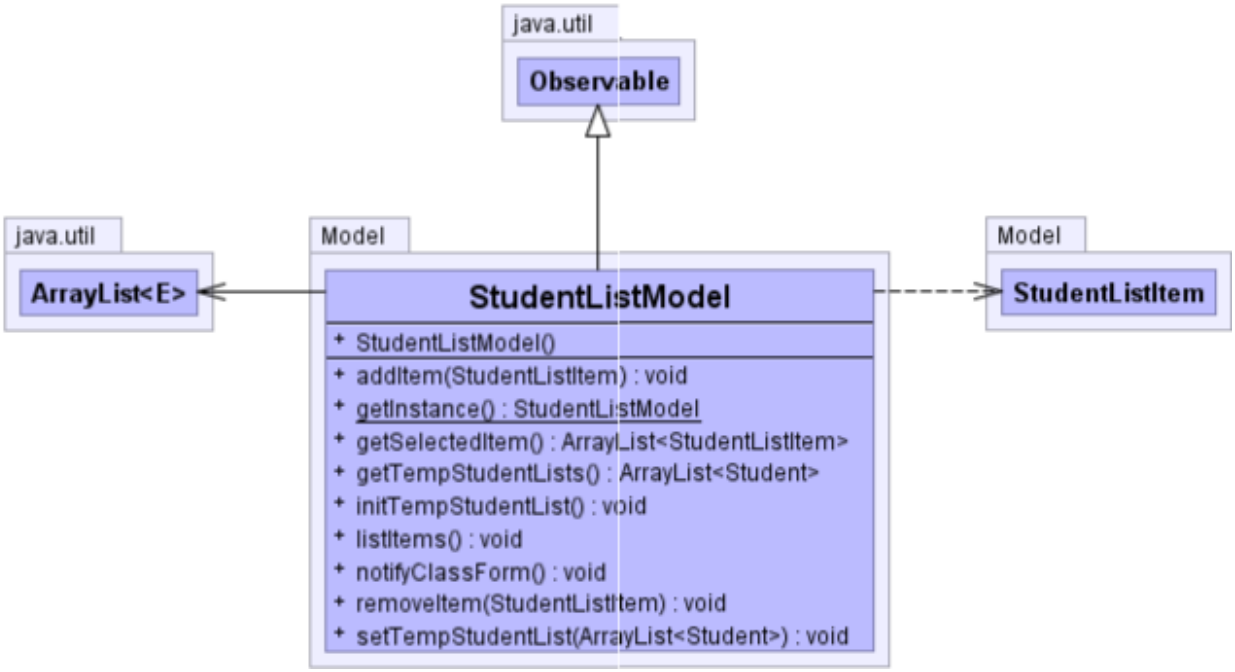


5.2.9 StudentListItem



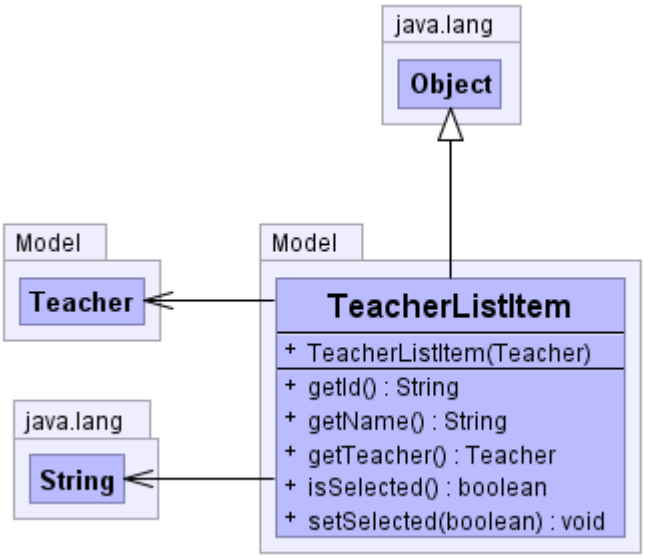
StudentListItem represents each student and used when selecting students to enroll in ClassForm. StudentListItem has a constructor that takes a Student parameter to get as many info about a student as possible. This class has properties such Id, name, isSelected (to know the state of each item in the list), and appropriate getters and setters.

5.2.10 StudentListModel



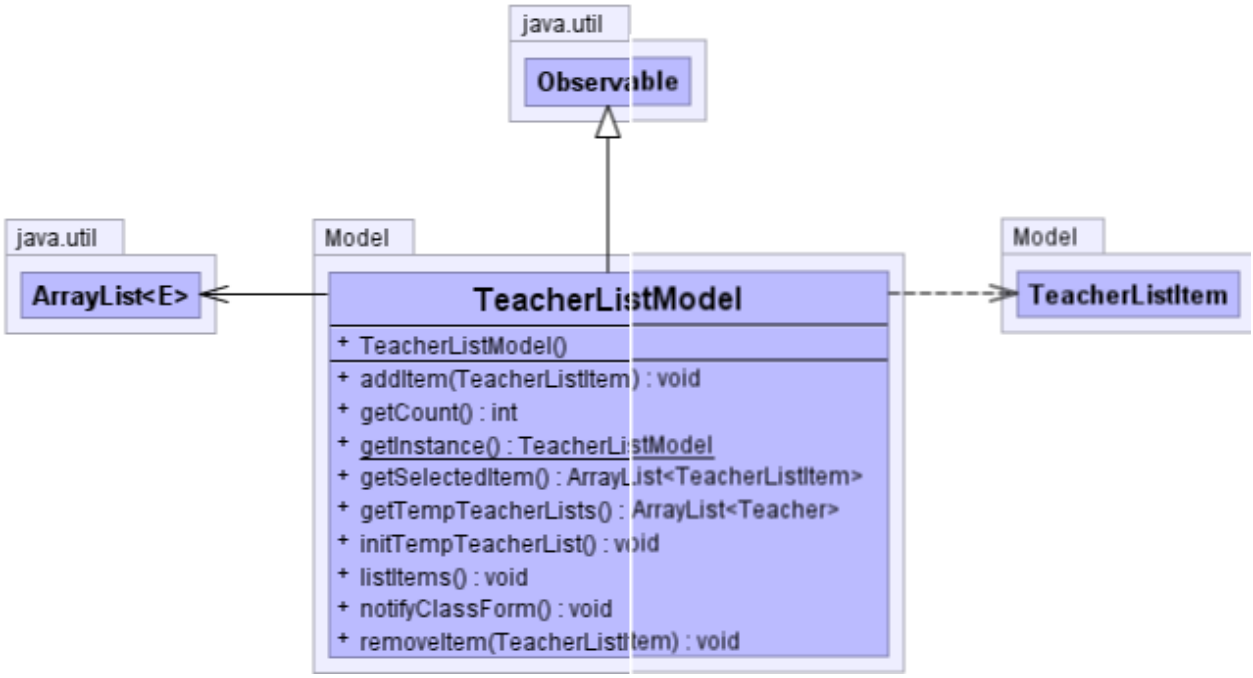
StudentListModel is used to store array list of selected StudentListItem in a student list. The selected items list will then be used in to associate changes for related classes. This class extends Observable to use setChanged() and notifyObservers() to tell ClassForm to update accordingly based on the number of students selected. setChanged() and notifyObservers() is put inside notifyClassForm() so that they can be called faster and at a same time.

5.2.11 TeacherListItem



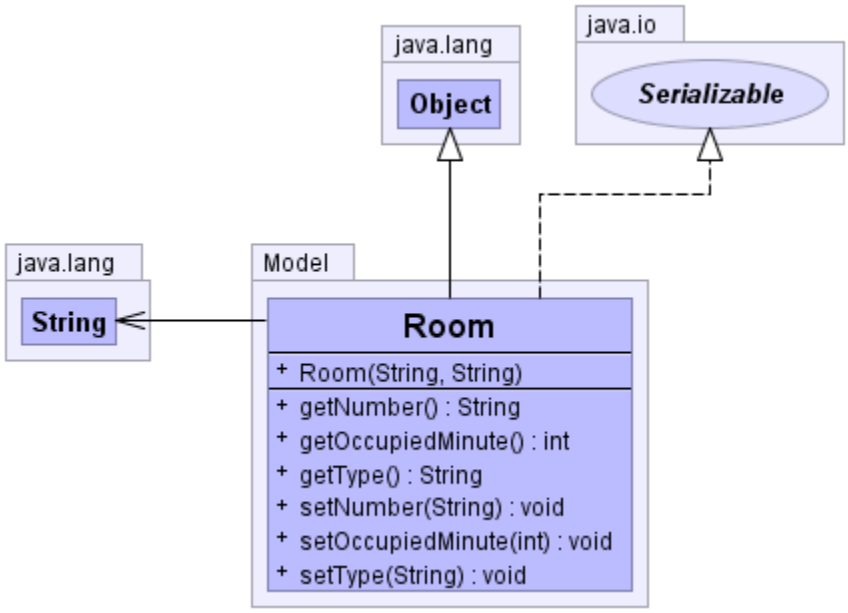
TeacherListItem represents each teacher and used when selecting teachers to assign into a class from ClassForm view. Similar to StudentListItem. TeacherListItem has a constructor that takes a Teacher parameter to get as many info about a teacher as possible. This class has properties such Id, name, isSelected (to know the state of each item in the list), and appropriate getters and setters.

5.2.12 TeacherListModel



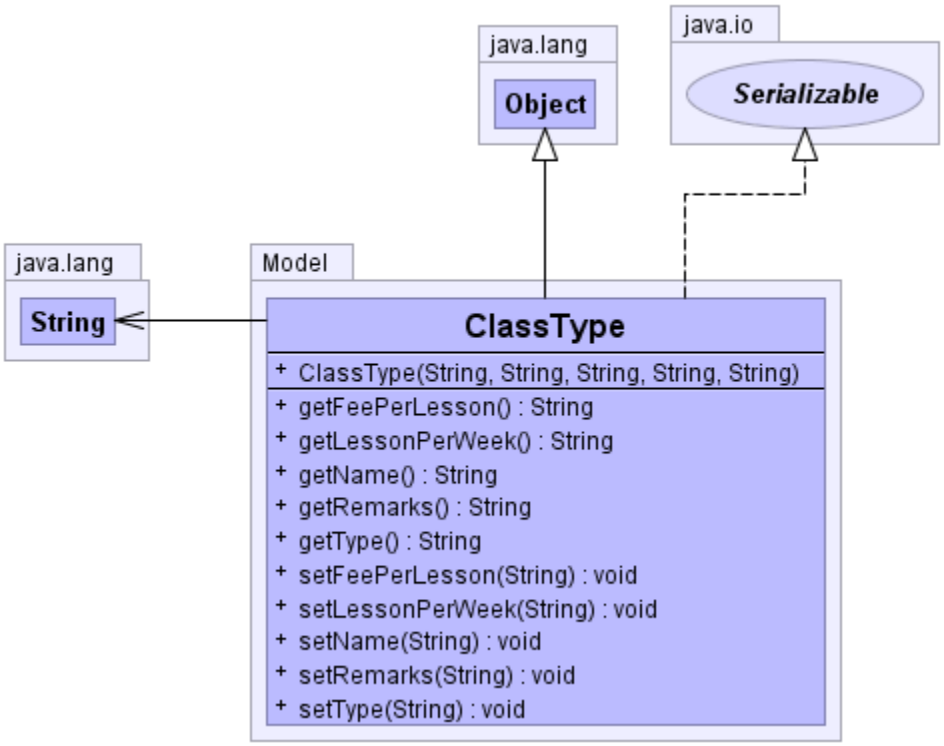
TeacherListModel is used to store array list of selected TeacherListItem in a teacher list. The selected items list will then be used in to associate changes for related classes. This class extends Observable to use setChanged() and notifyObservers() to tell ClassForm to update accordingly based on the number of students selected. setChanged() and notifyObservers() is put inside notifyClassForm() so that they can be called faster and at a same time.

5.2.13 Room



Room class contains properties to reflect an instance of a real class room like room number, and type. Room also contains a property called occupiedMinute to calculate utilization ratio. This class uses String type in constructors ,to initialize the properties when it is created, as well as in setters.

5.2.14 ClassType



Eache class should have a clas type and ClassType is used to represent class type of a class. ClassType also determines amount of fee per lesson and numer of lessons can be taught in a week. It has appropriate getters and setters to retrieve and set information.

5.3 Controller

This package contains the controller classes of the program. Each controller class is associated with a component in View package. Furthermore, Controller classes are responsible to call the accurate method from the model classes when an event is triggered in the views.


Controller
ActivateButtonController
GenerateFormController
AllTimeTableController
HistoryController
MobilePhoneController
AssignmentFormController
HomePhoneController
ReportChooseController
ChooseOptionsController
ImageFileView
ReportFormController
ClassFormController
ImageFilter
RoomFormController
ClassTypeController
ImagePreview
StaffFormController
ControlPanelFormController
ImportController
StudentFormController
CopyClassController
InvoiceFormController
StudentListController
DeleteButtonController
LanguageController
TeacherFormController
EnrollmentFormController
LoginFormController
TeacherListController
ExportController
ManagerFormController
Utils

5.4 View

Each class in View package is associated with other class in Model and Controller classes. The view displays data and fire event to the controller so that appropriate action can be taken to process/display the appropriate data.