Renumeration System

Architectural Pattern

An architectural pattern is a general, reusable solution to a commonly occurring problem in software architecture within a given context. Architectural patterns are similar to software design pattern but have a broader scope.

1. **Client-server pattern**
2. **Master-slave pattern**
3. **Pipe-filter pattern**
4. **Broker pattern**
5. **Peer-to-peer pattern**
6. **Event-bus pattern**
7. **Model-view-controller pattern**
8. **Blackboard pattern**
9. **Interpreter pattern**
10. **Layered Pattern**

Architectural Pattern that we are using for this Project is -:

Model-view-controller pattern

This pattern, also known as MVC pattern, divides an interactive application in to 3 parts as,

1. **model** — contains the core functionality and data
2. **view** — displays the information to the user (more than one view may be defined)
3. **controller** — handles the input from the user

This is done to separate internal representations of information from the ways information is presented to, and accepted from, the user. It decouples components and allows efficient code reuse.

Why Model-view-controller patter?

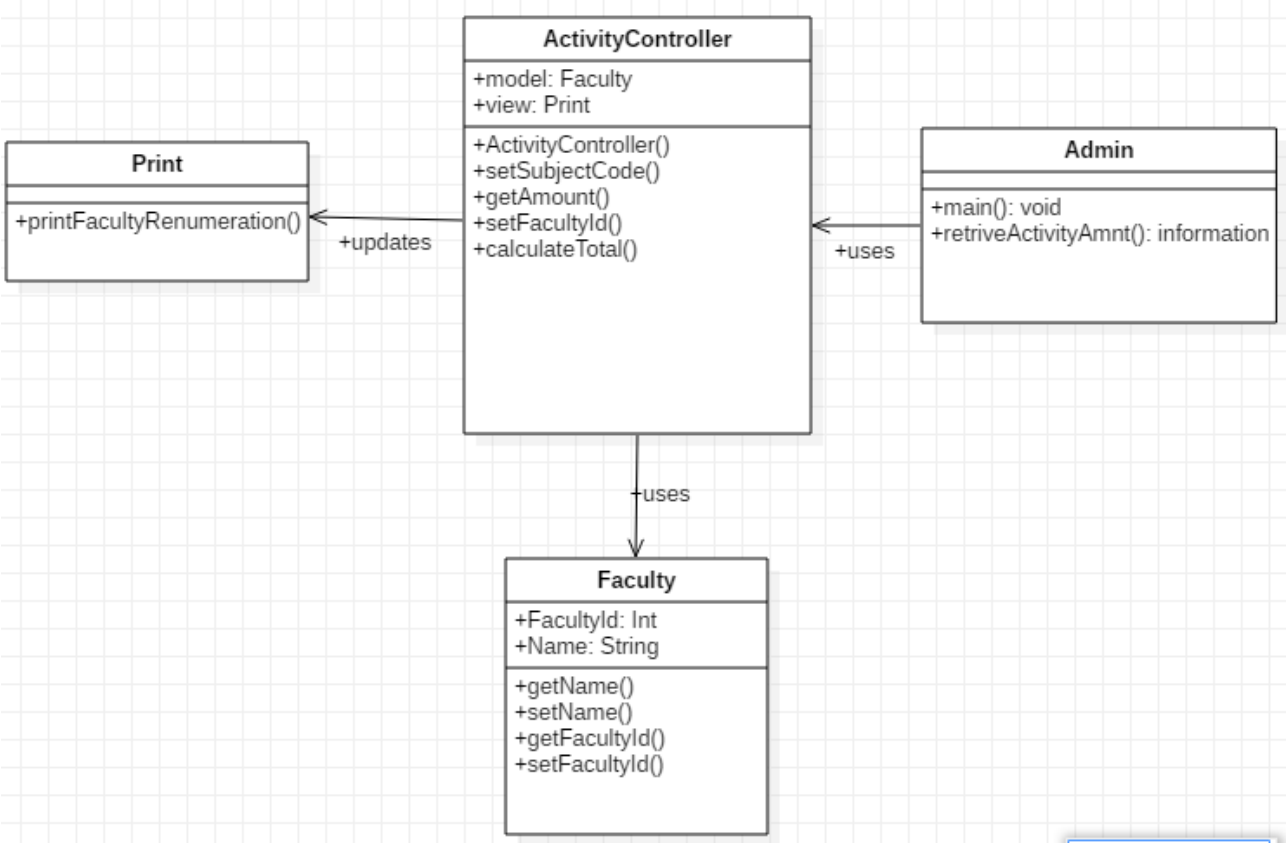
The Model is divided into 3 parts which is stated above as **MODEL,VIEW and CONTROLLER.** Since our project is loosely coupled,where each and every model has a seperate class of implementation so we decided using MVC model.

The **Model** (for example, the data information) contains only the pure application data; it contains no logic describing how to present the data to a user.

The **V*iew*** (for example, the presentation information) presents the model's data to the user. The view knows how to access the model's data, but it does not know what this data means or what the user can do to manipulate it.

Finally, the **Controller** (for example, the control information) exists between the view and the model. It listens to events triggered by the view (or another external source) and executes the appropriate reaction to these events. In most cases, the reaction is to call a method on the model. Since the view and the model are connected through a notification mechanism, the result of this action is then automatically reflected in the view.

OUR MVC ARCHITECTURAL PATTERN.



We are going to create a *Faculty* object acting as a model *Print* will be a view class which can print student details on console and *ActivityController* is the controller class responsible to store data in *Faculty* object and update view *Print* accordingly