# MODULE-1 “Linking All Class & Admin-Side”

### 1.1. DESCRIPTION OF THE MODULE:

A part of this module is linking all other moduels using the concept of oop’s (interface’s, abstract, inheritance, etc) so that the code will be more precise, and also each Page represents a GUI and a class. And all class’s have many things in common, so instead of writing the same thing again in each class, I have used interface and abstract-class’s to store everything in common-place, and all the other class extends this abstract and interface class’s. So using the concept of inheritance the child class can inherit all the methods, and variables are inherited. So in short a part of this module is to link all class’s. In this part of my module there are three class’s

***Interface (“InterfaceClass”):*** This is a interface class where I will store all the variables, methods etc. which are common to both the admin and teacher side, in a common place, so that I don’t need to redeclare again

***Abstract class (“TeacherStorage”):*** This is abstract class where I will store the things which are common only to the teacher-side class’s.

***Abstract class (“AdminStorage”):*** This is abstract class where I will store the things which are common only to the Admin-side class’s.

The other part in my module is the admin side class’s where, the admin is the head of all teachers he can access all teachers and he can fetch all their personal-information, their timetable and also message admin. So the full adminside is taken control by me. In short this class will accept or reject the pass that the teachers send’s to me, also the admin will recruit new teachers, with the credendetails the admin given to a teacher, the teacher can finish the verification process only with the help of the details given by the admin. So this module has a total of 5-class’s and all this class’s are related only to the admin side.

***Class (“AdminAllTeachersInfo”):*** This class will display all the details that the teacher has filled in the signup page, using a table. This class is basically to show all the info of the teachers in a common page.

***Class (“AdminHomePage”):*** This class is the home-page of the admin side class’s, so this class just mainly contains only a GUI-part of the admin side class.

***Class (“AdminNotification”):*** This class is a important part in our java project, where the admin recivies the notification from the teacher stating that they need of approval of the pass, so In this part admin will eithier approve or reject the pass, sended by that specific teacher. In short this class is for accepting the pass that the teacher has sended.

***Class (“AdminRecruitTeachers”):*** Suppose if there is a scenario where the admin recruit a new teacher, then that teacher will be given credendetails name, collegeID (unique) and password. So with the help of this details only a teacher can go-through the verification page.

***Class (“TeacherAdmin”):*** This is the place where the collegeID given to teachers that ID is entered by the admin, and now the admin can access all the details their personal-information, their timetable and also message admin etc.

Overall this module focus on linking all the other class’s in a common page, and path. And also all the class’s related to the admin side.

### 1.2. MOTIVATION OF THE MODULE:

The motivation behind developing this module was driven by many factors the main reason is to create a admin side and also to differeintiate the admin and teachers side. And the interface and abstract class make the code very opimized and save’s time and space also. So the main motivation of this module are the following goals

***Reusing Code:*** We wanted to avoid repeating code and make development more efficient. By using interfaces and abstract classes, we could store common elements, variables, and methods in one place. This way, we could easily reuse them in different parts of the project. It also made it easier to update or modify the code since changes made in one location would automatically apply to all related parts.

***Organized and Easy-to-Read Code:*** We aimed to make the code well-structured and easy to understand. Each page in the project represents a specific graphical interface and class. By using inheritance, the code became more organized and modular. It improved readability, making it easier for developers to navigate and comprehend the different parts of the project. This also made troubleshooting and maintenance simpler.

***Flexibility and Scalability:*** We wanted to ensure that the project could easily adapt to future changes and additions. By using interfaces and abstract classes, we created a flexible foundation. This allowed us to extend or modify functionalities without major code rewrites. New features and modules could be integrated smoothly by extending existing classes or implementing interfaces. This flexibility ensures that the project can grow and evolve over time.

***Efficient Admin-Side Management***: We focused on streamlining administrative tasks and making them easier to handle. The admin-side classes were designed to provide dedicated interfaces and functionalities for administrators. This empowered administrators to efficiently manage teacher-related information. They could access teacher details, handle notifications, approve or reject requests, and recruit new teachers through user-friendly interfaces. This streamlined the administrative process and reduced manual work.

In summary, the motivation behind Module-1 was to improve code reusability, enhance code organization and readability, support flexibility and scalability, and streamline administrative management. By linking classes and using object-oriented principles, the project became easier to maintain, expand, and handle administrative tasks efficiently.

## **1.3 RELEVANCE OF THE MODULE IN THE SYSTEM:**

The Module-1, which focuses on linking all classes and implementing the admin-side functionalities, is very important for the system. It plays a key role in managing the system effectively and making things easier for administrators. The relevance of this module can be seen in the following ways:

***Admin Control***: The module allows the system to have an administrator who oversees all the teachers. This helps in maintaining proper control and supervision of teacher-related activities within the system.

***Simplifying Admin Tasks:*** By providing specific classes for admins, the module makes administrative tasks simpler and more organized. Admins can easily access teacher information, manage notifications, approve or reject requests, and recruit new teachers using user-friendly interfaces. This reduces manual work and increases efficiency.

***Ensuring Secure Verification:*** The module ensures a secure process for verifying new teachers. Admins assign unique credentials like names, college IDs (unique), and passwords to each teacher. These credentials are necessary for teachers to complete the verification process. By managing the verification, admins ensure that only eligible and trustworthy teachers are recruited.

***Scalability:*** The module's use of object-oriented concepts, like interfaces, abstract classes, and inheritance, makes the system flexible and scalable. It becomes easier to add new features and modules to the system by extending existing classes or implementing interfaces. This flexibility allows the system to adapt to future needs and expand smoothly.

In summary, Module-1 is highly relevant as it establishes admin control, simplifies administrative tasks, ensures secure verification of teachers, and provides scalability to the system. By connecting classes and implementing admin-side functionalities, this module greatly contributes to the smooth and efficient management of the entire system.

## **1.4. UML OF THE MODULE**

### 1.4.1. UML of “AdminAllTeachersInfo” Class.

### 

**Figure 1** UML of AdminAllTeachersInfo Class.

### 1.4.2. UML Of “AdminHome” Class.

### 

**Figure 2** UML Of “AdminHome” Class.

### 1.4.3. UML Of AdminNotification Class.

### 

### *Figure 3 UML Of AdminNotification Class.*

### 1.4.4. UML Of AdminRecruitTeachers Class.

### 

### *Figure 4 UML Of AdminRecruitTeachers Class.*

### 1.4.5. UML Of AdminStorage Abstract Class.

### 

**Figure 5** AdminStorage Abstract Class.

### 1.4.6. UML Of TeacherAdmin Class.

### 

**Figure 6** TeacherAdmin Class.

### 1.4.7. UML Of InterfaceClass interface Class.

### 

**Figure 7** UML Of InterfaceClass interface Class.

### 1.4.8. UML Of TeacherStorage Abstract Class.

### 

**Figure 8** UML Of TeacherStorage Abstract Class.

### 1.4.9. UML Of AdminStorage Abstract Class.

### 

**Figure 9** UML Of AdminStorage Abstract Class.

## 1.5. OUTPUT OF THE MODULE

### 1.5.1. AdminHeader

## 

**Figure 10** Output Image Of AdminHeader.

## **1.5.2. AdminFooter**

## 

**Figure 11** Output Image Of AdminFoote.

### 1.5.3.TeacherFooter

## 

**Figure 12** Output Image Of TeacherFooter.

### 1.5.4.TeacherHeader

## 

**Figure 13** Output Image Of TeacherHeader.

### 1.5.5. AdminAllTeachersInfo

## 

**Figure 14** Output Image Of AdminAllTeachersInfo.

### 1.5.6. AdminHome

## 

**Figure 15** Output Image Of AdminHome.

### 1.5.7. AdminNotification

## 

**Figure 16** Output Image Of AdminNotification.

### 1.5.8. AdminRecruitTeachers

## 

**Figure 17** Output Image Of AdminRecruitTeachers.