Assignment 8
Identification and Implementation of GRASP pattern

AIM - Apply any four GRASP patterns to refine the Design Model for a given problem description. Using effective UML 2 diagrams and implementing them with a suitable object-oriented language.

PROBLEM STATEMENT -

1) Identification and Implementation of GRASP pattern.
2) Apply any two GRASP patterns to refine the Design Model for a given problem description.
3) Using effective UML 2 diagrams and implementing them with a suitable object-oriented language.

OBJECTIVE -

1) To Study GRASP patterns.
2) To implement a system using any two GRASP Patterns.

THEORY-

General Responsibility Assignment Software Patterns, abbreviated GRASP, consist of guidelines for assigning responsibility to classes and objects in object-oriented design . It is not related to the SOLID design principle. All patterns answer some software problems, and these problems are common to almost every software development project. These techniques have not been invented to create new ways of working, but to better document and standardize old, tried-and-tested programming principles in object-oriented design.

Following are the pattern in GRASPS :

1) **Controller**

   The Controller is responsible for handling the requests of actors. The Controller is the middle-man between your user clicking “Send” and your back-end making that happen. The Controller knows how to interpret the action of user-interfaces, and how to connect those actions to behaviors in your System.
2) **Creator**

   The Creator takes the responsibility of creating certain other objects.

3) **Indirection**

   The indirection pattern supports low coupling and reuses potential between two elements by assigning the responsibility of mediation between them to an intermediate object. An example of this is the introduction of a controller component for mediation between data (model) and its representation (view) in the model-view control pattern. This ensures that coupling between them remains low.

4) Information expert

The Expert Pattern solves this by encapsulating information about a task into a distinct class.

5) Low coupling

Low coupling is an evaluative pattern that dictates how to assign responsibilities for the following benefits: lower dependency between the classes, change in one class having a lower impact on other classes, higher reuse potential.

6) High cohesion

Very similar to Low Coupling – Often related (but not always) – Should be considered in every design decision.

● Lower cohesion almost always means: – An element more difficult to understand, maintain, or reuse – An element more likely to be affected by the change.

● Low cohesion suggests that more delegation should be used.

7) Polymorphism

With respect to implementation, this usually means the use of a super (parent) class or interface – Coding to an interface is generally preferred and avoids committing to a particular class hierarchy.

● Code like the following should raise a red flag! Switch creatureType Case batType: print "Screech!" Case cowType: print "Moooooo..." Case humanType: print "Let's watch TV!" [...]

● Also see GRASP pattern #8, Protected Variations.

8) Protected variations

The protected variations pattern protects elements from the variations on other elements (objects, systems, subsystems) by wrapping the focus of instability with an interface and using polymorphism to create various implementations of this interface.

9) Pure fabrication

A pure fabrication is a class that does not represent a concept in the problem domain, specially made up to achieve low coupling, high cohesion, and the reuse potential thereof derived.

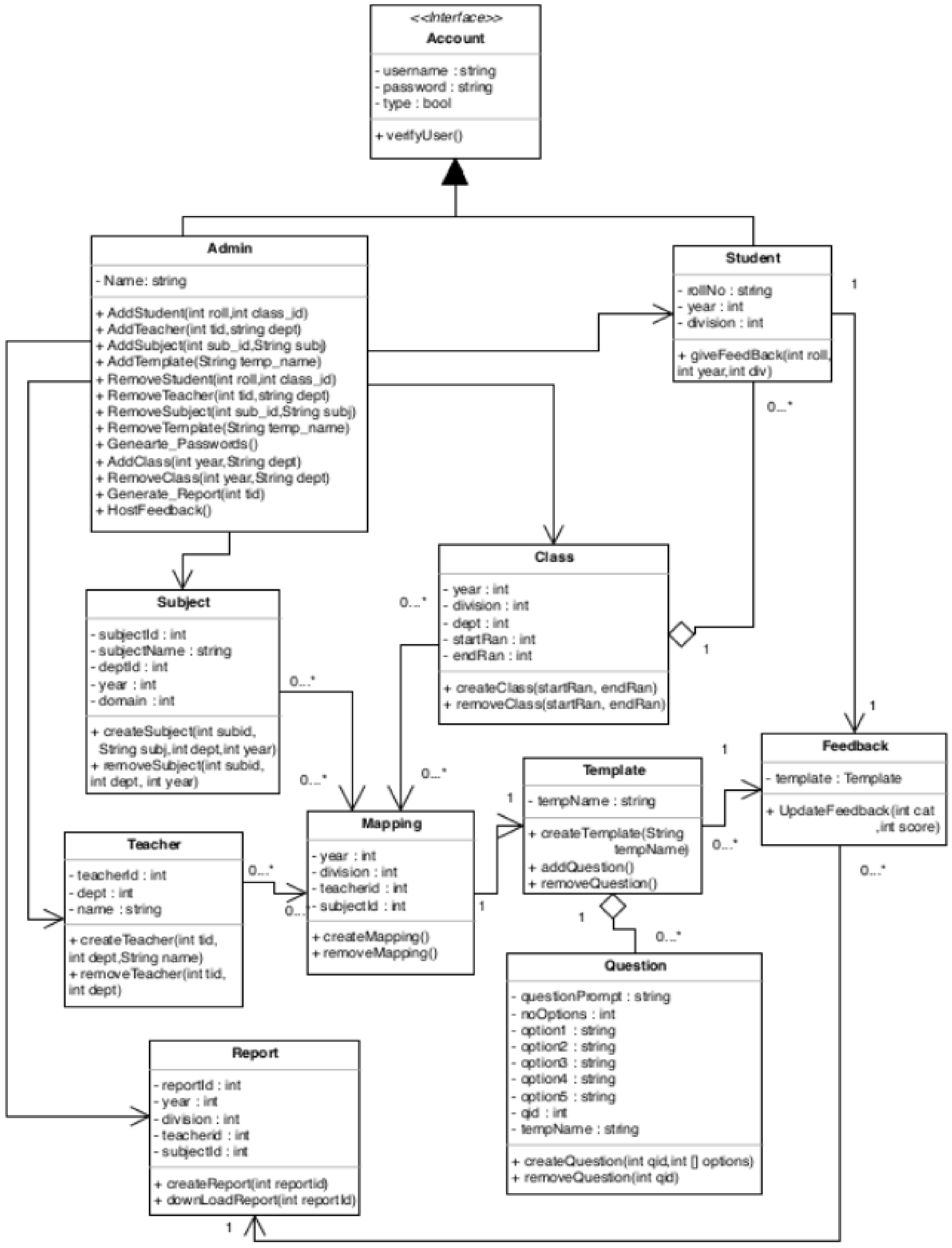


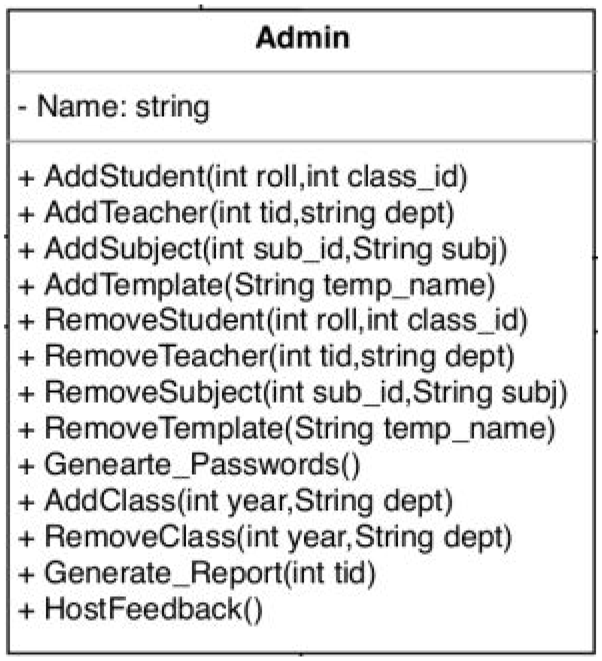
Figure 1: Class Diagram

Implementation

1) **CONTROLLER**
   Problem:
   What the first object beyond the UI layer receives and coordinates "controls" a system operation?

   Solution:
   Assign the responsibility to an object representing one of these choices:
   – Represents the overall "system", "root object", device that the software is running within, or a major subsystem (these are all variations of a facade controller)
   – Represents a use case scenario within which the system operation occurs (a use case or session controller)

   This principle implementation depends on the high-level design of our system but generally, we need to always define objects which orchestrate our business transaction processing. At first glance, it would seem that the MVC Controller in Web applications/APIs is a great example here (even the name is the same) but for me, it is not true. Of course, it receives input but it shouldn't coordinate a system operation – it should delegate it to a separate service or Command Handler:


Admin
- Name: string
+ AddStudent(int roll,int class_id)
+ AddTeacher(int tid,string dept)
+ AddSubject(int sub_id,String subj)
+ AddTemplate(String temp_name)
+ RemoveStudent(int roll,int class_id)
+ RemoveTeacher(int tid,string dept)
+ RemoveSubject(int sub_id,String subj)
+ RemoveTemplate(String temp_name)
+ Genearte_Passwords()
+ AddClass(int year,String dept)
+ RemoveClass(int year,String dept)
+ Generate_Report(int tid)
+ HostFeedback()

Admin class is the controller object.

- It represents the overall "system" or "root object" that the software is running within or a major subsystem
- Admin class is used to handle and connect most other objects.
- It controls objects like Student, Teacher, Subject, Template, Class and Report.

- Admin objects are responsible to interact with users and accordingly add, modify, or delete the data of the system.
- It is also responsible to carry out some of the crucial operations of the system such as Generate_Passwords and Generate_Reports

Admin Object controls the User actions right from the beginning of the system. After Login, users need to fill in the data as well as host the feedback. After that Students will give the respective feedback for faculty and then reports are available to view. All of these major actions are controlled by the Admin object

Admin controls Class:
The admin object is responsible for the creation of the class. But after creation, there are some other major activities that the class object performs. These activities are
Getting mapped with the teacher and subject
Create students according to the assigned roll number
Modify any of the contents with respect to class.

Admin controls Subject:
The admin object is responsible for the creation of the Subject. But after creation, there are some other major activities that the subject-object performs. These activities are
Getting mapped with the teacher and class
Modify any of the contents with respect to the subject.

Admin controls Teacher:
The admin object is responsible for the creation of the Teacher. But after creation, there are some other major activities that the teacher object performs. These activities are Getting mapped with the subject and class.
Modify any of the contents with respect to the teacher.

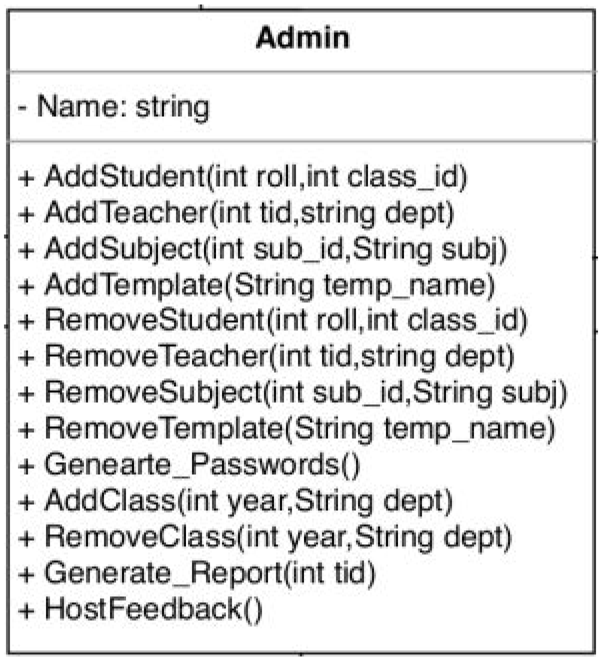
Admin controls Question:
The admin object is responsible for the creation of the Questions and also Templates. But after creation, there are some other major activities that the Question object performs. These activities are
Getting mapped with the respective template
Getting mapped with the teacher class subject mapping
Modify any of the contents with respect to the question and its template.

Admin controls Other Major operations:
The admin object is responsible for the creation of the Report and Passwords. But after creation, there are some other major activities that the Report object performs. These activities are
Generating reports
Creating passwords
Hosting Feedback

Modify any of the contents with respect to reports.

2) **CREATOR**
   Problem:
   Who should be responsible for creating a new instance of some class?

   Solution:
   The creation of objects is one of the most common activities in an object-oriented system. Which class is responsible for creating objects is a fundamental property of the relationship between objects of particular classes.
   The creator is a GRASP Pattern which helps to decide which class should be responsible for creating a new instance of a class. Object creation is an important process, and it is useful to have a principle in deciding who should create an instance of a class.



Admin
- Name: string
+ AddStudent(int roll,int class_id)
+ AddTeacher(int tid,string dept)
+ AddSubject(int sub_id,String subj)
+ AddTemplate(String temp_name)
+ RemoveStudent(int roll,int class_id)
+ RemoveTeacher(int tid,string dept)
+ RemoveSubject(int sub_id,String subj)
+ RemoveTemplate(String temp_name)
+ Genearte_Passwords()
+ AddClass(int year,String dept)
+ RemoveClass(int year,String dept)
+ Generate_Report(int tid)
+ HostFeedback()

Admin class is the creator object.

Admin Creates Class:

- The admin object is responsible for the creation of the class.
- Create students according to the assigned roll number.

Admin Creates Subject:

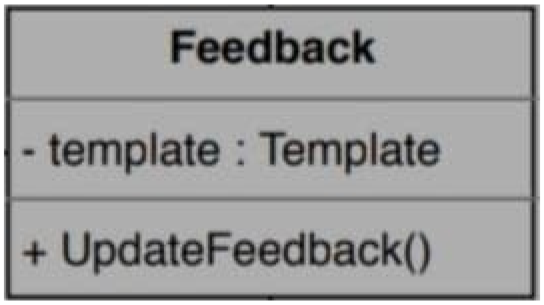
- The admin object is responsible for the creation of the Subject.
- Creation of any of the contents with respect to the subject.

Admin Creates Report:

- The admin object is responsible for the creation of the Report.
- Creation of any of the reports depending on the type chosen.

3) Information Expert
Problem:
What is a general principle of assigning responsibilities to objects?

Solution:
Assign responsibility to the information expert - the class that has the information necessary to fulfill the responsibility.
Information expert (also expert or the expert principle) is a principle used to determine where to delegate responsibilities such as methods, computed fields, and so on. Using the principle of information expert, a general approach to assigning responsibilities is to look at a given responsibility, determine the information needed to fulfill it, and then determine where that information is stored. This will lead to placing the responsibility on the class with the most information required to fulfill it.



Feedback class is the information expert object.

Feedback holds responsibility for maintaining feedback:
- The Feedback object is responsible for storing feedback from the student.
- Students according to the assigned roll number give feedback whose report is generated.

Feedback is responsible for student wise record:
- The Feedback object is responsible for each mapped Subject, Teacher, Student.
- Feedback of any of the contents with respect to each of the mappings.

- **Class Structure in this project:**

```
// Super class Account (Base class of Student and Admin)
class Account {
   // Name, Email and Password
   private String username, password;

   // Initialize name and email to NULL
   Account() {
      name = null;
         password = null;
   }

   // Methods to get Name, Email and Password
   public String getName() {
      /* Get Name */
      return null;
   }

   public String getPassword() {
      /* Get Password */
      return null;
   }

   // Protected methods to set Name, Email and password provided
   protected void setName(String username) {
      /* Set Name */
   }

   protected void setPassword(String password) {
      /* Set password */
   }

   private boolean isValid(String username, String password) {
      if(quantity == NULL || password == NULL)
         return false;
      return true;
   }

   public Boolean verifyUser(String username, String password) {
         if (isValid(username, password))
      {
         this.username = username;
```

```
                        this.password = password;
                        return true;
        }
        else
        {
            // Error
                return false;
        }

}

class Admin extends Account {
    // Attribute of Admin
    private String name;

    // Get name of admin
    public String getName() {
        return null;
    }

    public void addStudent(int roll,int class_id) {
        /* add details of student */
    }

    public void addTeacher(int tid,String dept) {
        /* add details of teacher */
    }

    public void addSubject(int sub_id,String subj) {
        /* add subjects */
    }

    public void addTemplate(String temp_name) {
        /* add template of questions */
    }

    public void removeStudent(int roll,int class_id)     {
            /* remove student */
    }

    public void removeTeacher(int tid,String dept)     {
            /* remove teacher */
    }
    public void removeSubject(int sub_id,String subj) {
            /* remove subject */
    }

    public void removeTemplate(String temp_name)  {
            /* remove template */
    }

    public void generatePasswords() {
        /* generate passwords */
    }

    protected void addClass(int year,String dept) {
        /* add class  */
    }

    public void removeClass(int year,String dept)       {
            /* remove class */
    }

    public void generateReport() {
        /* generate passwords */
    }
}
```

CONCLUSION -

Thus in this assignment, we have successfully identified and implemented GRASP patterns.