Shri Ramdeobaba College of Engineering and Management Nagpur

Name- Harsh Agrawal CSE SEM-6 Roll no-43 Section-B

Subject-Software Engineering LAB.

Practical no- 4

Aim- Creating a Class Diagram To Implement the Employee Management System

---

Theory->

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system classes, their attributes, operations (or methods), and the relationships among objects.

Purpose of Class Diagrams

1. Shows static structure of classifiers in a system
2. Diagram provides basic notation for other structure diagrams prescribed by UML
3. Helpful for developers and other team members too
4. Business Analysts can use class diagrams to model systems from business perspective

A UML class diagram is made up of: A set of classes and a set of relationships between classes.

What is a Class

A description of a group of objects all with similar roles in the system, which consists of:

- Structural features (attributes) define what objects of the class.
- Represent the state of an object of the class
- Are descriptions of the structural or static features of a class
- Behavioral features (operations) define what objects of the class.
- Define the way in which objects may interact
- Operations are descriptions of behavioral or dynamic features of a class

Class Notation

A class notation consists of three parts:

1. Class Name

The name of the class appears in the first partition.

2. Class Attributes

Attributes are shown in the second partition.

The attribute type is shown after the colon.

Attributes map onto member variables (data members) in code.

3. Class Operations (Methods)

- Operations are shown in the third partition. They are services the class provides.
- The return type of a method is shown after the colon at the end of the method signature.
- The return type of method parameters are shown after the colon following the parameter name.
- Operations map onto class methods in code

Class diagram is a static diagram and it is used to model the static view of a system. The static view describes the vocabulary of the system.

Class diagrams are also considered as the foundation for component and deployment diagrams. Class diagrams are not only used to visualize the static view of the system but they are also used to construct the executable code for forward and reverse engineering of any system.

Generally, UML diagrams are not directly mapped with any object-oriented programming languages but the class diagram is an exception.

Class diagram clearly shows the mapping with object-oriented languages such as Java, C++, etc. From practical experience, class diagrams are generally used for construction purposes.

In a nutshell it can be said, class diagrams are used for −

- Describing the static view of the system.
- Showing the collaboration among the elements of the static view.
- Describing the functionalities performed by the system.
- Construction of software applications using object oriented languages.

In UML, a relationship is a connection between model elements. A UML relationship is a type of model element that adds semantics to a model by defining the structure and behavior between model elements. UML relationships are grouped into the following categories:

Category	**Function**
Activity edges	Represent the flow between activities
Associations	Indicate that instances of one model element are connected to instances of another model element
Dependencies	Indicate that a change to one model element can affect another model element
Generalizations	Indicate that one model element is a specialization of another model element

Realizations	Indicate that one model element provides a specification that another model element implements
Transitions	Represent changes in state

The following topics describe the relationships that you can use in class diagrams:

- **Abstraction relationships**
  An abstraction relationship is a dependency between model elements that represents the same concept at different levels of abstraction or from different viewpoints. You can add abstraction relationships to a model in several diagrams, including use-case, class, and component diagrams.
- **Aggregation relationships**
  In UML models, an aggregation relationship shows a classifier as a part of or subordinate to another classifier.
- **Association relationships**
  In UML models, an association is a relationship between two classifiers, such as classes or use cases, that describes the reasons for the relationship and the rules that govern the relationship.
- **Association classes**
  In UML diagrams, an association class is a class that is part of an association relationship between two other classes.
- **Binding relationships**
  In UML models, a binding relationship is a relationship that assigns values to template parameters and generates a new model element from the template.
- **Composition association relationships**
  A composition association relationship represents a whole–part relationship and is a form of aggregation. A composition association relationship specifies that the lifetime of the part classifier is dependent on the lifetime of the whole classifier.
- **Dependency relationships**
  In UML, a dependency relationship is a relationship in which one element, the client, uses or depends on another element, the supplier. You can use dependency relationships in class diagrams, component diagrams, deployment diagrams, and use-case diagrams to indicate that a change to the supplier might require a change to the client.
- **Directed association relationships**
  In UML models, directed association relationships are associations that are navigable in only one direction.
- **Element import relationships**
  In UML diagrams, an element import relationship identifies a model element in another package, and allows the element in the other package to be referenced by using its name without a qualifier.

- **Generalization relationships**
  In UML modeling, a generalization relationship is a relationship in which one model element (the child) is based on another model element (the parent). Generalization relationships are used in class, component, deployment, and use-case diagrams to indicate that the child receives all of the attributes, operations, and relationships that are defined in the parent.
- **Interface realization relationships**
  In UML diagrams, an interface realization relationship is a specialized type of implementation relationship between a classifier and a provided interface. The interface realization relationship specifies that the realizing classifier must conform to the contract that the provided interface specifies.
- **Instantiation relationships**
  In UML diagrams, an instantiation relationship is a type of usage dependency between classifiers that indicates that the operations in one classifier create instances of the other classifier.
- **Package import relationship**
  In UML diagrams, a package import relationship allows other namespaces to use unqualified names to refer to package members.
- **Realization relationships**
  In UML modeling, a realization relationship is a relationship between two model elements, in which one model element (the client) realizes the behavior that the other model element (the supplier) specifies. Several clients can realize the behavior of a single supplier. You can use realization relationships in class diagrams and component diagrams.
- **Usage relationships**
  In UML modeling, a usage relationship is a type of dependency relationship in which one model element (the client) requires another model element (the supplier) for full implementation or operation.

Class Diagram Employee Management System->

- This is the class diagram created for the Employee Management System. It contains a total of 14 classes which are Employee, Manager, System user, Super Admin,Account,Communication, SAAccount, SUAccount, MAccount,Leave, Task, Group, Attendance and Salary.
- Here is the brief description of all the classes.
- In the use case diagram for the employee management system there were four actors named Employee, Manager, System User, Super Admin.

- There are four different classes created for all the four different Actors.These classes have a direct association with the Account class. All the Actors have basic Account structure of account such as username, password etc and also they have several other features of their own.
- The Account class has a generalization with the communication class. This feature is created for all the users of the application where they can communicate with each other and share data with each other.
- All the four actors have four classes for their features and other four classes for their common basic entities such as name, dob etc. These classes have an association among them.

- **Class Employee->**

Attributes-> empid, ename, edob and eadd.

Functions->createAccount(), updateAccount()

- **Class Manager->**

Attributes-> mid, mname, mdob and madd.

Functions->createAccount(), updateAccount(),createEmployee()

- **Class SystemUser->**

Attributes-> suid, sname, sdob and sadd.

Functions->createAccount(), updateAccount()

- **Class SuperAdmin->**

Attributes-> said, saname, sadob and sadd.

Functions->createAccount(), updateAccount(), createManager(), createEmployee()

- **Class Account->**

Attributes-> uname,upass,upp,status,meetid

Functions->changeUsername(), changePassword(), changePP(), login(), logout(), scheduleMeetings(), joinMeeting(), communicate()

- **Class SAAccount->**

Attributes->mid,eid,uid,salid,

Functions->addManager(), removeManager(),addEmployee(), removeEmplyee(),manageSalary(), manageUser()

- **Class SUAccount->**

Attributes->

Functions->manageExperience(), manageLeave(), manageLogin(),addPeople(), removePeople()

- **Class MAccount->**

Attributes->

Functions->manageAttendence(), manageSalary(), manageLeave(), manageTask(), createGroup()

- **Class EAccount->**

Attributes->eaid, eatype,

Functions->checkAttendence(), checkSalary(), applyLeave(), joinGroup, checkTask()

- **Class Leave->**

Attributes->leave

Functions->applyLeave(), cancelLeave(), approveLeave()

- **Class Group->**

Attributes-> gid, gname, gtype

Functions->joinGroup(), leaveGroup()

- ❖ **Class Task->**

Attributes->taskid, taskName, taskType,taskStatus

Functions->addTask(), removeTask(),updateTask(), generateReport(),manageTask()

- ❖ **Class Salary->**

Attributes->sal

Functions->checkSal(), addSal()

- ❖ **Class Attendance->**

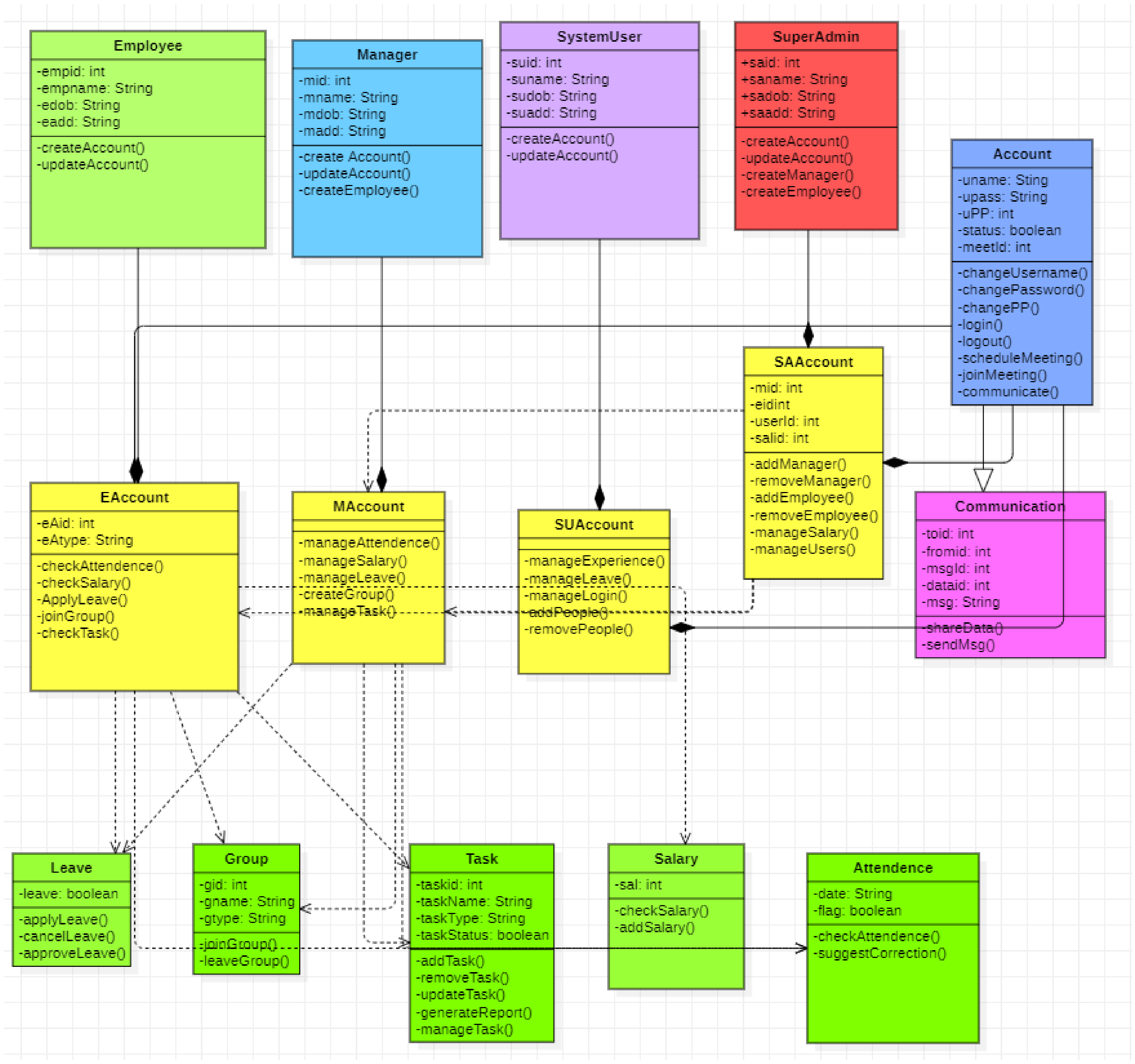
Attributes->date, flag

Functions->checkAttedence(), sugestCorrection()

- There are again four classes with the name such as Leave,Task, Group, Attendance and Salary.
- The leave class contains information for the Leave where the Employees and the managers can apply for the leave. The Approval for the leave is decided by the higher authority. For ex the manager can decide whether to grant or reject a leave asked by the employee.
- Before asking for a particular feature the user has to check if he/she can get access to a particular feature. Hence check feature condition is applied.
- Now we also have a class named Group. Employees can only join or leave the group. The Group creation task can be only performed by the Managers and The Super Admin.
- We Also have a class named Task. The Managers and Super Admin can generate and remove tasks. Employees can only provide the update to the task which is assigned to them. There is also a flag with datatype boolean which can mark if the task is completed or not.
- There is a class named salary where the managers and super Admin can add the salary and Employees can only check the salary.

- There is also a class Attendance were the managers can manually keep the record of the Attendance, can add, delete and update attendance. Employees can only view their attendance and suggest the corrections if any.
- Hence this is the Class Diagram for the demonstration for the Employee Management System.

<u>Class Diagram for Employee Management System-></u>



Result- Class Diagram for Employee Management has been designed and code conversion has been demonstrated Successfully !!

The End!!