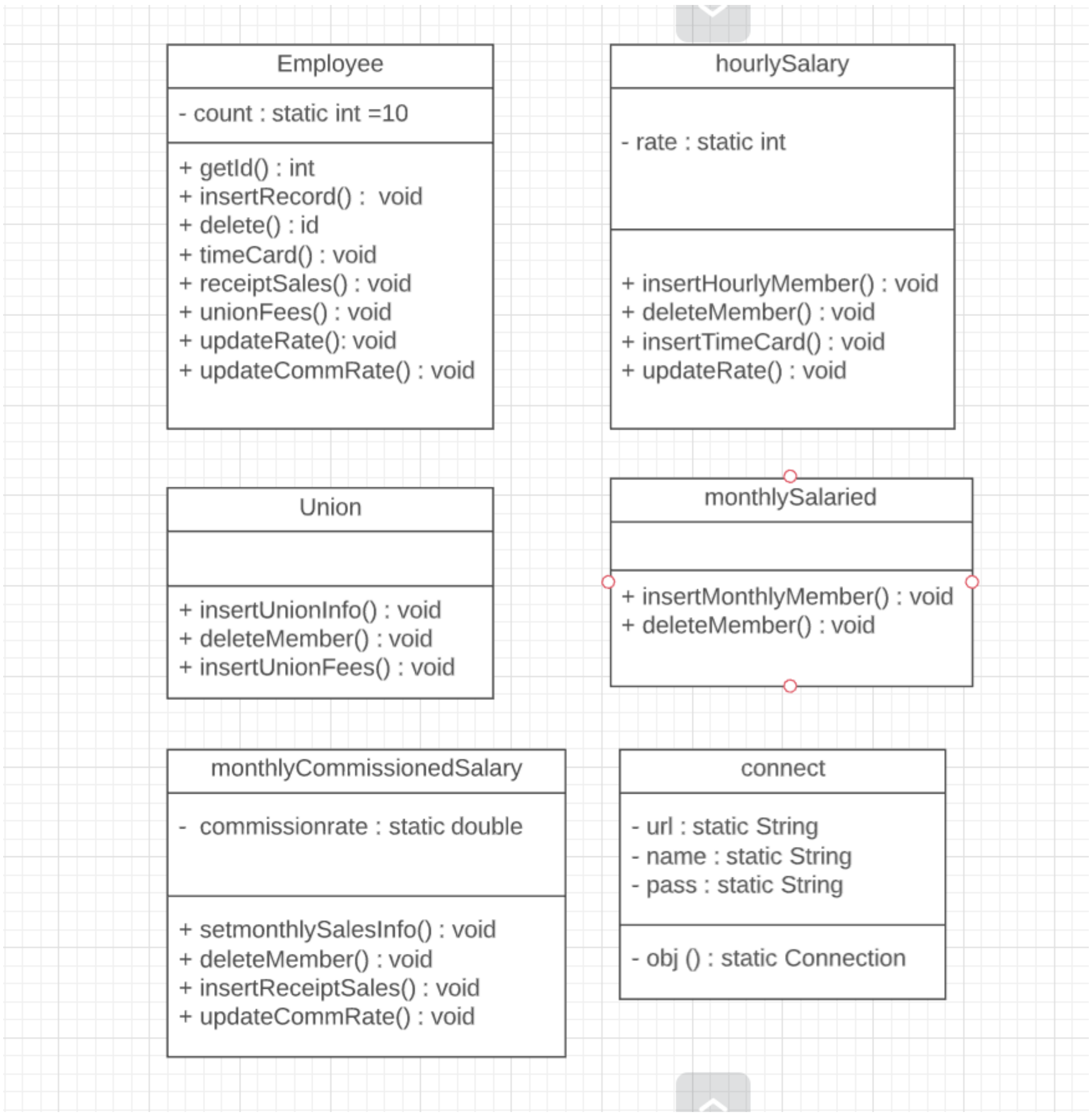
UML Class Daigram



Assumptions

1. I have assumed that we are taking input from a database which contains 5 tables with the following columns.

   employee – id, name, address, umember, paymentmethod, lastpaid

   hourly – Id, rate, hrs, date

   union – Id, servicecharge, date

   monthly – Id, salary

   monthlySales – Id, date, amountsales, commissionrate, salary

2. I have assumed the default rate for Hourly employees to be 10% and default rate for monthly commissioned employees to be 5.5% .

3. I have made assumed that the input for sales receipt for monthly commissioned employees and union service charge for union members is given on every Sunday.

4. For deleting an employee record the user must know the ID of the employee.

5. User can only change the rate for hourly employees and Commission rate for monthlySales Employees.

6. The sales receipt and union service charge are updated every Sunday

Design Choices

1. I created 7 classes in total – main, Employee, hourlySalary, monthlySalaried, Union, Connect and monthlyCommissionedSalary .

    Employee – Interacts with all the employees

    Union – Interacts with employees who are part of Union

    hourlySalary – Interacts with employees who are paid weekly on for the number of hours worked

    monthlySalaried – Interacts with employees who are paid monthly

    monthlyCommissionedSalary – Interacts with employees who are paid monthly as well on the basis of sales.

    Connect – Helps to creates a link between local database and java IDE with the help of JDBC.

2. When creating the Union,hourlySalary, monthlySalaried,monthlyCommissionedSalary classes we could have used inheritance with Employee as the base class. But that would have only added unnecessary variables and also inheritance makes the code messy. So I created all these classes separately.
3. Took all the input in main only instead of directly inserting into the table. This helps to improve code maintainability as the work is happening at one place.
4. In order to prevent the user from interacting directly with the Union,hourlySalary, monthlySalaried,monthlyCommissionedSalary classes I created another class Employee. The user interacts with the Employee class only. This helps us to prevent the user from coming in contact with sensitive data.
5. Delete function is called from the main class based on user's choice. Instead of creating separate functions in Employee class to check fir the Id in every table we can use a single delete function (of Employee Class) which fetches the record for that employee from the Employee table

and then with the help of umember and paymentmethod variable we can make the call for delete only in those class where it actually exist.

6. Based on the above delete method I improved my Add function by making a single insertRecord function in employee which then call all the required functions where the record for he current employee should be present.
7. Point 5 and 6 help to reduce unnecessary functions, reduce the code size and improve its maintainability and also help to implement SOLID principles.

Before :

```
Public void setUnionInfo(int id){
        Union u=new Union();
        u.insertRecord(id);
}
Public void setHourlyInfo(int id){
        hourly u=new hourly();
        u.insertRecord(id);
}
Public void setMonthlyInfo(int id,int salary){
        monthlySalaried u=new monthlySalaried ();
        u.insertRecord(id,salary);
}

Public void setMonthlySalesInfo(int id,int salary){
        monthlyCommissionedSalary u=new monthlyCommissionedSalary();
        u.insertRecord(id,salary);
}
```

```
Main(){
…
        Employee E=new Employee();
        if(Umember==1){
                E.setUnionInfo(id)
        }
        if(paymentMethod==1){
                E.setHourlyInfo(id)
        }
        else if(paymentMethod==2){
                E.setMonthlyInfo(id,salary)
        }
        else{
                E.setMonthlySalesInfo(id,salary)
        }
…
}
```

After :

```
public void insertRecord(int Id,String Name,String Address,int Umember,int
paymentRecieveMethod,int paymentMethod,String lastpaid,int salary) throws Exception
{
                if(Umember==1){
                        Union u=new Union();
                        u.insertUnionInfo(Id);
                }
                if(paymentMethod==1){
                        hourlySalary h=new hourlySalary();
                        h.insertHourlyInfo(Id);
                }
                else if(paymentMethod==2){
                        monthlySalaried m=new monthlySalaried();
                        m.insertMonthlyInfo(Id,salary);
                }
                else{
                        monthlyCommissionedSalary ms=new monthlyCommissionedSalary();
                        ms.setMonthlySalesInfo(Id,salary);
                }
                ..
}

Main(){
…
E.insertRecord(Id,Name,Address,Umember,paymentRecieveMethod,paymentMethod,salary);
…
}
```

Reduced it to single function for inserting values in any of the necessary table.

8. In order to implement abstraction I have used a single functions of employee class for implementing all the functionalities.
9. Not declared Employee as interface because then giving every employee a unique ID becomes a problem. We cannot use object count for giving employee ID as all member variables declared in interfaces are final by default.