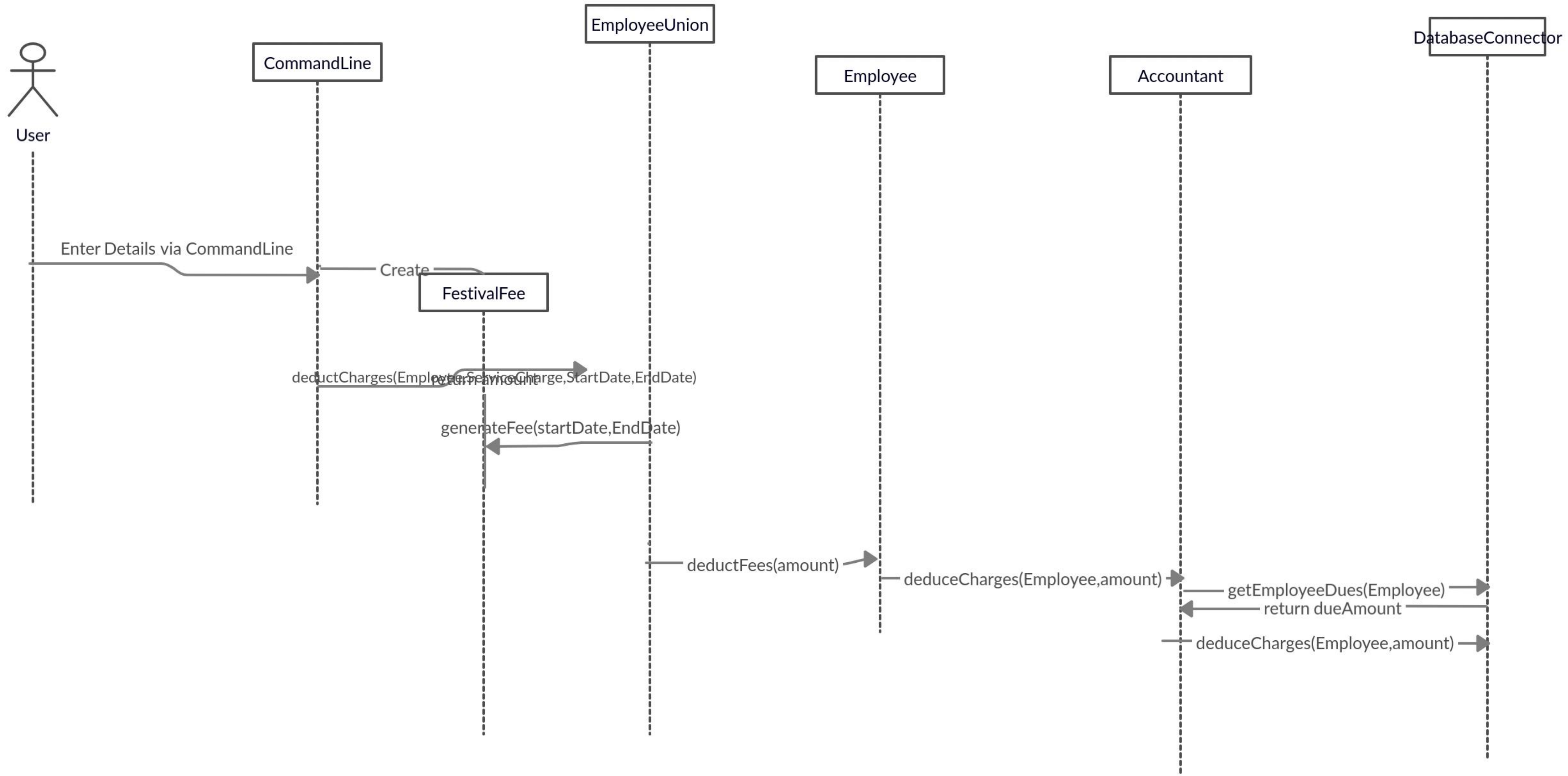
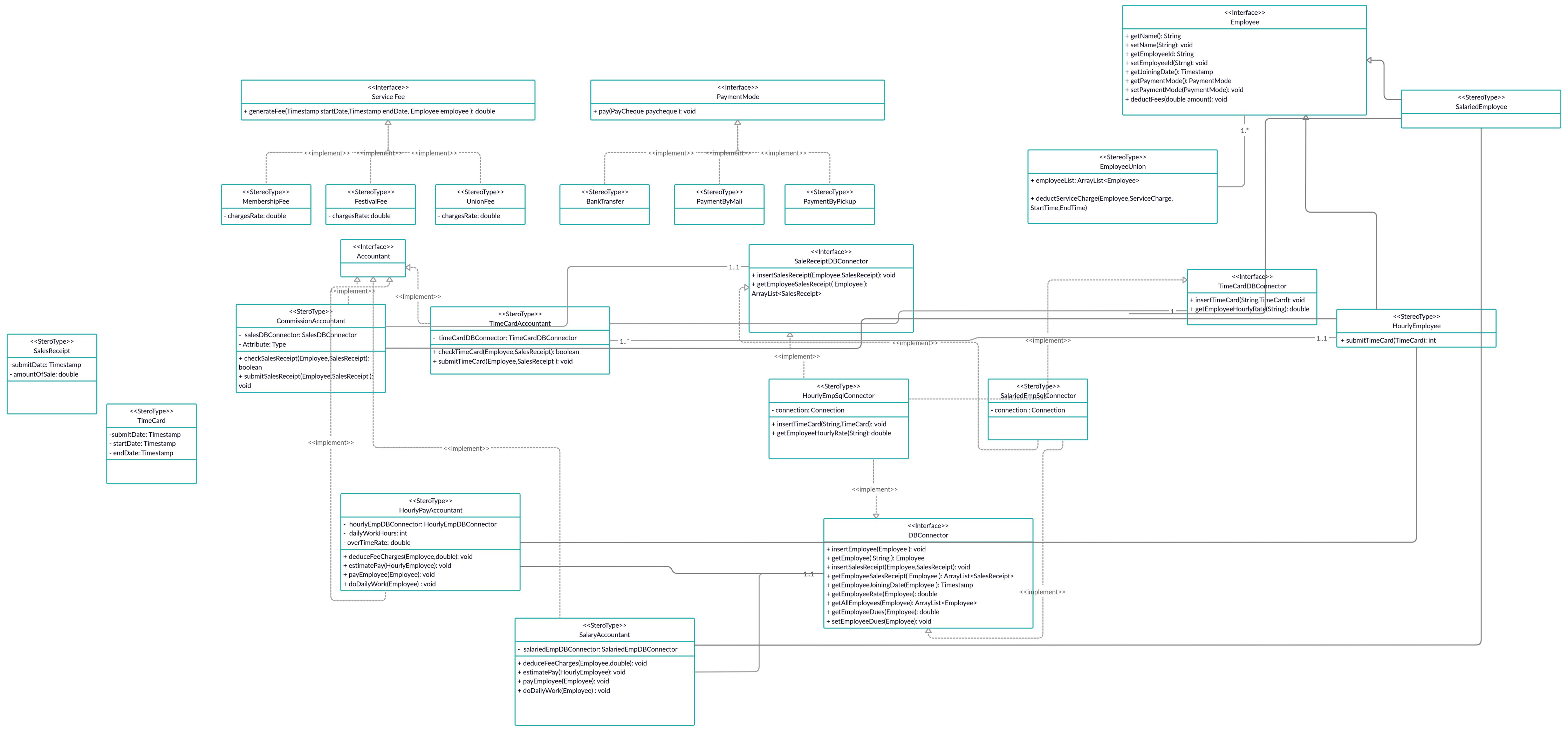

User
CommandLine
EmployeeUnion
Employee
Accountant
DatabaseConnector
Enter Details via CommandLine
Create
FestivalFee
deductCharges(Employee,ServiceCharge,StartDate,EndDate)
return amount
generateFee(startDate,EndDate)
deductFees(amount)
deduceCharges(Employee,amount)
getEmployeeDues(Employee)
return dueAmount
deduceCharges(Employee,amount)



<<Interface>>
Employee
+ getName(): String
+ setName(String): void
+ getEmployeeId: String
+ setEmployeeId(Strng): void
+ getJoiningDate(): Timestamp
+ getPaymentMode(): PaymentMode
+ setPaymentMode(PaymentMode): void
+ deductFees(double amount): void
<<Interface>>
Service Fee
+ generateFee(Timestamp startDate,Timestamp endDate, Employee employee): double
<<Interface>>
PaymentMode
+ pay(PayCheque paycheque): void
<<StereoType>>
SalariedEmployee
1.*
<<StereoType>>
EmployeeUnion
+ employeeList: ArrayList<Employee>
+ deductServiceCharge(Employee,ServiceCharge, StartTime,EndTime)
<<implement>>
<<implement>>
<<implement>>
<<implement>>
<<implement>>
<<implement>>
<<StereoType>>
MembershipFee
- chargesRate: double
<<StereoType>>
FestivalFee
- chargesRate: double
<<StereoType>>
UnionFee
- chargesRate: double
<<StereoType>>
BankTransfer
<<StereoType>>
PaymentByMail
<<StereoType>>
PaymentByPickup
<<Interface>>
Accountant
<<Interface>>
SaleReceiptDBConnector
+ insertSalesReceipt(Employee,SalesReceipt): void
+ getEmployeeSalesReceipt(Employee): ArrayList<SalesReceipt>
1..1
<<Interface>>
TimeCardDBConnector
+ insertTimeCard(String,TimeCard): void
+ getEmployeeHourlyRate(String): double
<<implement>>
<<implement>>
<<SteroType>>
CommissionAccountant
- salesDBConnector: SalesDBConnector
- Attribute: Type
+ checkSalesReceipt(Employee,SalesReceipt): boolean
+ submitSalesReceipt(Employee,SalesReceipt): void
<<SteroType>>
TimeCardAccountant
- timeCardDBConnector: TimeCardDBConnector
+ checkTimeCard(Employee,SalesReceipt): boolean
+ submitTimeCard(Employee,SalesReceipt): void
1..*
1
<<StereoType>>
HourlyEmployee
+ submitTimeCard(TimeCard): int
1..1
<<implement>>
<<implement>>
<<SteroType>>
SalesReceipt
-submitDate: Timestamp
- amountOfSale: double
<<StereoType>>
TimeCard
-submitDate: Timestamp
- startDate: Timestamp
- endDate: Timestamp
<<implement>>
<<SteroType>>
HourlyEmpSqlConnector
- connection: Connection
+ insertTimeCard(String,TimeCard): void
+ getEmployeeHourlyRate(String): double
<<SteroType>>
SalariedEmpSqlConnector
- connection : Connection
<<implement>>
<<implement>>
<<implement>>
<<SteroType>>
HourlyPayAccountant
- hourlyEmpDBConnector: HourlyEmpDBConnector
- dailyWorkHours: int
- overTimeRate: double
+ deduceFeeCharges(Employee,double): void
+ estimatePay(HourlyEmployee): void
+ payEmployee(Employee): void
+ doDailyWork(Employee) : void
<<Interface>>
DBConnector
+ insertEmployee(Employee): void
+ getEmployee(String): Employee
+ insertSalesReceipt(Employee,SalesReceipt): void
+ getEmployeeSalesReceipt(Employee): ArrayList<SalesReceipt>
+ getEmployeeJoiningDate(Employee): Timestamp
+ getEmployeeRate(Employee): double
+ getAllEmployees(Employee): ArrayList<Employee>
+ getEmployeeDues(Employee): double
+ setEmployeeDues(Employee): void
1..1
<<implement>>
<<SteroType>>
SalaryAccountant
- salariedEmpDBConnector: SalariedEmpDBConnector
+ deduceFeeCharges(Employee,double): void
+ estimatePay(HourlyEmployee): void
+ payEmployee(Employee): void
+ doDailyWork(Employee) : void

Design Objectives and Approaches

Design objective is to design the Payroll management system.

Approach to the problem is by having problem divided into layers as follows

> ***GUI Layer***
> ***Employee Layer***
> ***Accountant Layer***
> ***Database Layer***

Each Layer is tried to be independent from below Layers as much as possible with the help of interface exposed by each layer.

Design Role and Responsibilities with Reasoning

GUI Layer - This consists of App (main file) class which was included very late to the project as the project is tried to be as decoupled as possible. It takes input from command line and output at command line.

Employee Layer - This layer consists of classes such as different types of employees, employee union, Service charges and its types, Time Card and SalesReceipt Classes. These classes generally are used as data storage or for their own use according to their type. Employee tell their accountant to serve some purpose.

Accountant Layer - This layer is analogical to accountant in offices which input data into database. Same work is done by different accountant classes at this layer. They also help in computing the salary, charges and input this into the database with the help of Database Layer.

Database Layer - This layer can have different implementation and database connectivity programs which can be changed anytime to support flexibility and decoupling in the system.

Future Design Improvements, Current Design Challenges and Alternative Design

Currently, design looks good but due to lack of time improper interfaces have been exposed at upper layers which increases coupling between classes.

This coupling can be reduced by improving database design and making proper changes in its interface and to the upper layers.

Current Challenge
Currently EmployeeID initial letter is used to determine the type of employee which increases the conditions introduced in code. This in turn increased the coupling of Employee Layer with Database layer.

Better Design

Introduce EmployeeType Registration

There should be a framework or contract through which we should always register a new employee type to a EmployeeTypeRegister class which would hold all necessary information required to get Correct instance of an Employee using only the employeeId.