DESIGN DOCUMENT – v1.0

by Harshit Mahajan

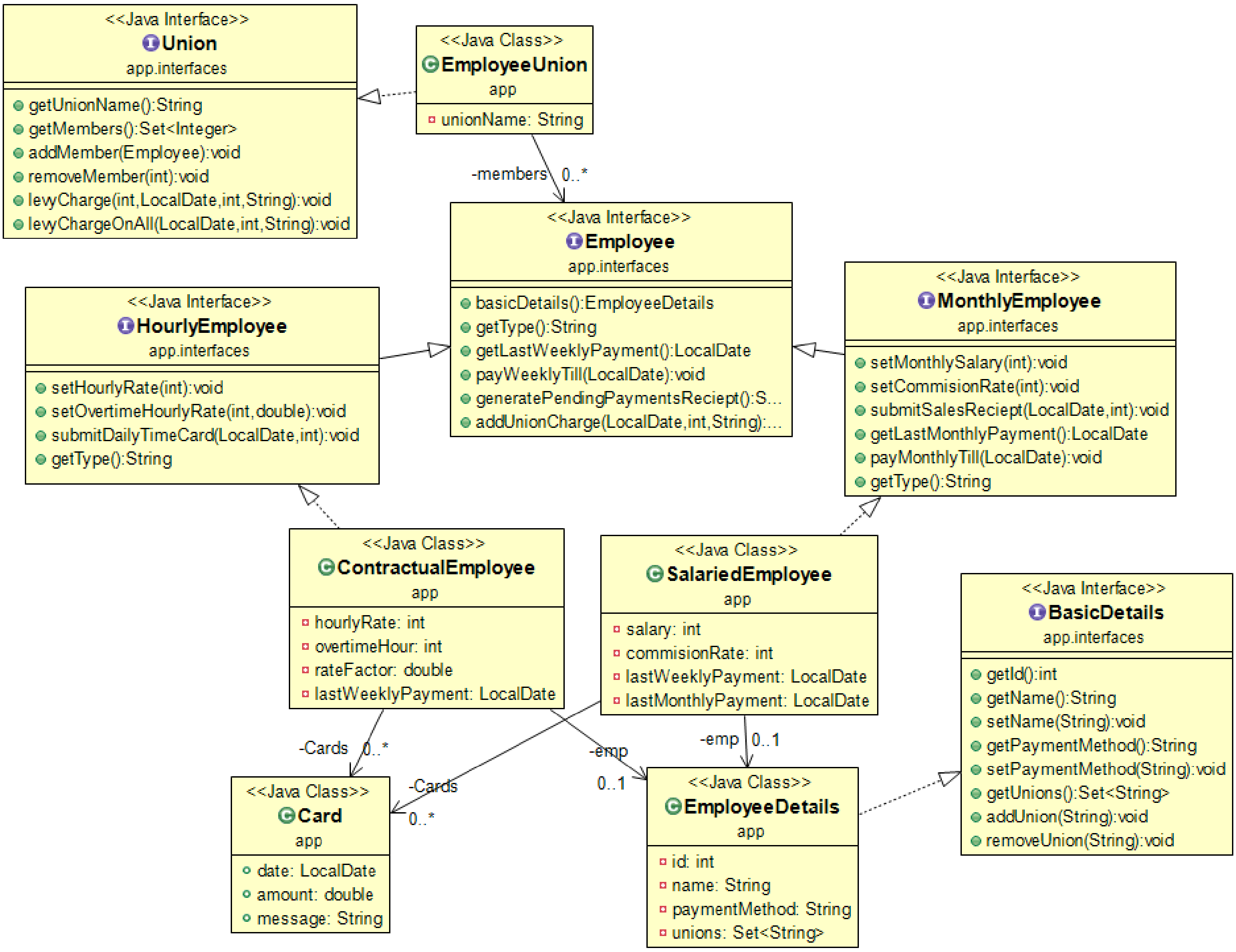
Class Diagrams

Employee Type:

- All Employee types must implement Employee interface.
- No class implements Employee interface, it provides the basic skeleton for an Employee.
- Any Employee type must have a EmployeeDetails object (has-a relationship i.e composition) which can be retrieved using basicDetails()
- An employee can be a member of any number of unions.

Union Type:

- A union can have any number of member Employees.
- A union can levy charges on one specific member or on all members.

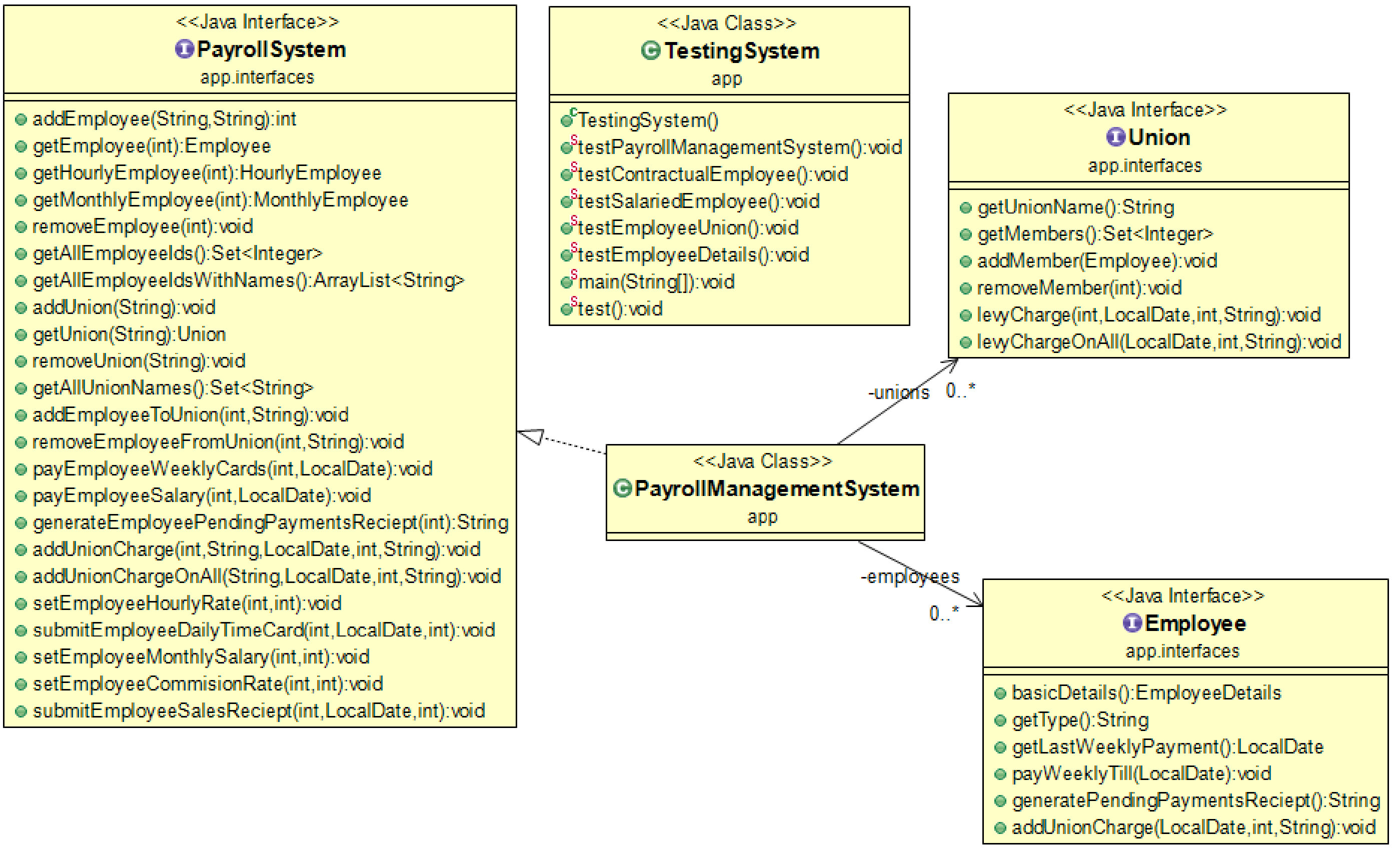


Payroll Management System:

- Stores a list of employees and unions.
- Provides a high level API for all operations, which can be plugged easily into a UI.

Testing System:

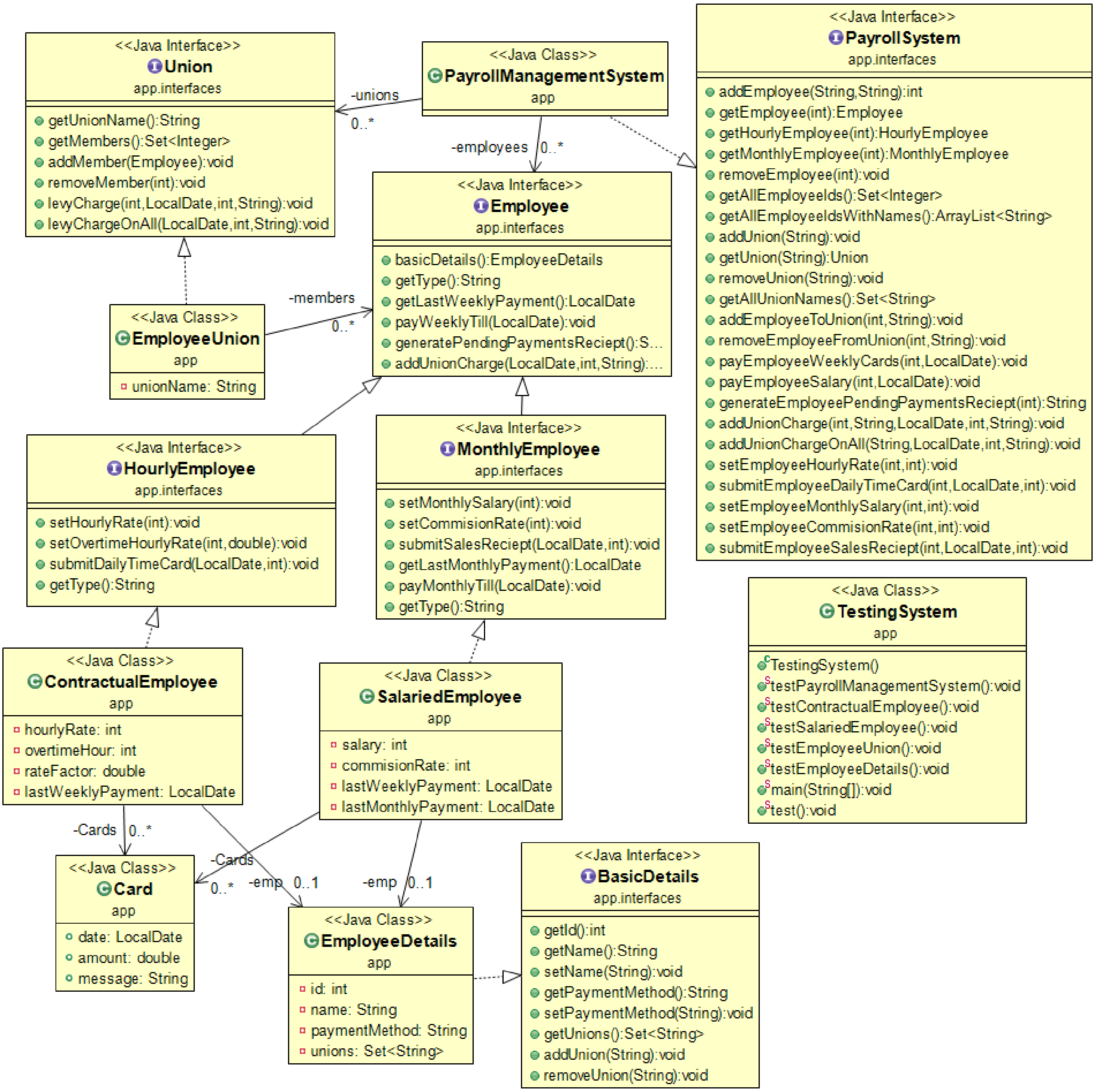
- Provides static functions which can be used for testing individual classes.



<<Java Interface>>
PayrollSystem
app.interfaces
addEmployee(String,String):int
getEmployee(int):Employee
getHourlyEmployee(int):HourlyEmployee
getMonthlyEmployee(int):MonthlyEmployee
removeEmployee(int):void
getAllEmployeeIds():Set<Integer>
getAllEmployeeIdsWithNames():ArrayList<String>
addUnion(String):void
getUnion(String):Union
removeUnion(String):void
getAllUnionNames():Set<String>
addEmployeeToUnion(int,String):void
removeEmployeeFromUnion(int,String):void
payEmployeeWeeklyCards(int,LocalDate):void
payEmployeeSalary(int,LocalDate):void
generateEmployeePendingPaymentsReciept(int):String
addUnionCharge(int,String,LocalDate,int,String):void
addUnionChargeOnAll(String,LocalDate,int,String):void
setEmployeeHourlyRate(int,int):void
submitEmployeeDailyTimeCard(int,LocalDate,int):void
setEmployeeMonthlySalary(int,int):void
setEmployeeCommisionRate(int,int):void
submitEmployeeSalesReciept(int,LocalDate,int):void
<<Java Class>>
TestingSystem
app
TestingSystem()
testPayrollManagementSystem():void
testContractualEmployee():void
testSalariedEmployee():void
testEmployeeUnion():void
testEmployeeDetails():void
main(String[]):void
test():void
<<Java Interface>>
Union
app.interfaces
getUnionName():String
getMembers():Set<Integer>
addMember(Employee):void
removeMember(int):void
levyCharge(int,LocalDate,int,String):void
levyChargeOnAll(LocalDate,int,String):void
-unions
0..*
<<Java Class>>
PayrollManagementSystem
app
-employees
0..*
<<Java Interface>>
Employee
app.interfaces
basicDetails():EmployeeDetails
getType():String
getLastWeeklyPayment():LocalDate
payWeeklyTill(LocalDate):void
generatePendingPaymentsReciept():String
addUnionCharge(LocalDate,int,String):void

Overview of the Whole System

- All classes are stored in package *app* in *SourceCode/src/app*
- All interfaces are stored in package *app.interfaces*
- All output *.class* files go in *SourceCode/bin*



<<Java Interface>>
Union
app.interfaces
getUnionName():String
getMembers():Set<Integer>
addMember(Employee):void
removeMember(int):void
levyCharge(int,LocalDate,int,String):void
levyChargeOnAll(LocalDate,int,String):void
-unions
0..*
<<Java Class>>
PayrollManagementSystem
app
-employees
0..*
<<Java Interface>>
PayrollSystem
app.interfaces
addEmployee(String,String):int
getEmployee(int):Employee
getHourlyEmployee(int):HourlyEmployee
getMonthlyEmployee(int):MonthlyEmployee
removeEmployee(int):void
getAllEmployeeIds():Set<Integer>
getAllEmployeeIdsWithNames():ArrayList<String>
addUnion(String):void
getUnion(String):Union
removeUnion(String):void
getAllUnionNames():Set<String>
addEmployeeToUnion(int,String):void
removeEmployeeFromUnion(int,String):void
payEmployeeWeeklyCards(int,LocalDate):void
payEmployeeSalary(int,LocalDate):void
generateEmployeePendingPaymentsReciept(int):String
addUnionCharge(int,String,LocalDate,int,String):void
addUnionChargeOnAll(String,LocalDate,int,String):void
setEmployeeHourlyRate(int,int):void
submitEmployeeDailyTimeCard(int,LocalDate,int):void
setEmployeeMonthlySalary(int,int):void
setEmployeeCommisionRate(int,int):void
submitEmployeeSalesReciept(int,LocalDate,int):void
<<Java Interface>>
Employee
app.interfaces
basicDetails():EmployeeDetails
getType():String
getLastWeeklyPayment():LocalDate
payWeeklyTill(LocalDate):void
generatePendingPaymentsReciept():S...
addUnionCharge(LocalDate,int,String):...
-members
0..*
<<Java Class>>
EmployeeUnion
app
unionName: String
<<Java Interface>>
HourlyEmployee
app.interfaces
setHourlyRate(int):void
setOvertimeHourlyRate(int,double):void
submitDailyTimeCard(LocalDate,int):void
getType():String
<<Java Interface>>
MonthlyEmployee
app.interfaces
setMonthlySalary(int):void
setCommisionRate(int):void
submitSalesReciept(LocalDate,int):void
getLastMonthlyPayment():LocalDate
payMonthlyTill(LocalDate):void
getType():String
<<Java Class>>
TestingSystem
app
TestingSystem()
testPayrollManagementSystem():void
testContractualEmployee():void
testSalariedEmployee():void
testEmployeeUnion():void
testEmployeeDetails():void
main(String[]):void
test():void
<<Java Class>>
ContractualEmployee
app
hourlyRate: int
overtimeHour: int
rateFactor: double
lastWeeklyPayment: LocalDate
<<Java Class>>
SalariedEmployee
app
salary: int
commisionRate: int
lastWeeklyPayment: LocalDate
lastMonthlyPayment: LocalDate
-Cards
0..*
-Cards
0..*
-emp
0..1
-emp
0..1
<<Java Class>>
Card
app
date: LocalDate
amount: double
message: String
<<Java Class>>
EmployeeDetails
app
id: int
name: String
paymentMethod: String
unions: Set<String>
<<Java Interface>>
BasicDetails
app.interfaces
getId():int
getName():String
setName(String):void
getPaymentMethod():String
setPaymentMethod(String):void
getUnions():Set<String>
addUnion(String):void
removeUnion(String):void

Design Decisions

Design Towards Loose Coupling

- Composition has been preferred over inheritance wherever possible.
- Eg: Employee Interface guarantees that each Employee must have a EmployeeDetails object, instead of say the Employee inheriting from a EmployeeDetails class. I was using inheritance at first, which lead to a lot of changes propagating to its subclasses whenever I changed the EmployeeDetails class.

Design Towards Abstract Types & Design By Contract

- System has been design using an interface first policy. Firstly an interface is defined for a given requirment.
- Firstly the behaviours/operations are defined, followed by the state information.

Design keeping Testing in mind

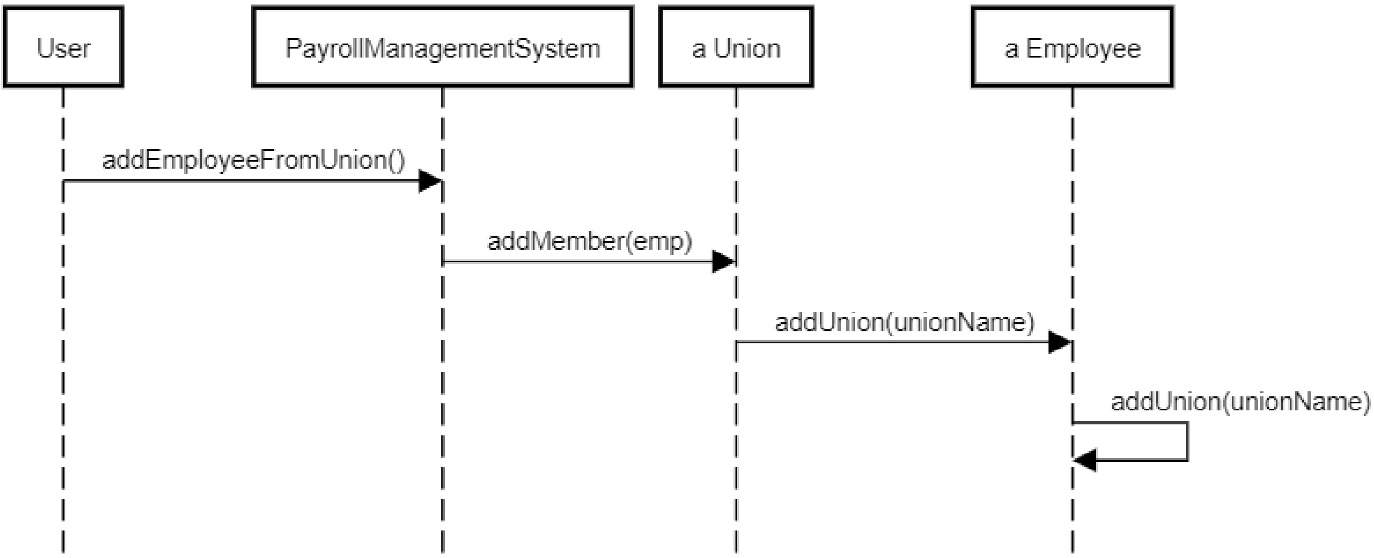
- The testing system exposes functions to test each class.

Design For Changeability and Future Requirements

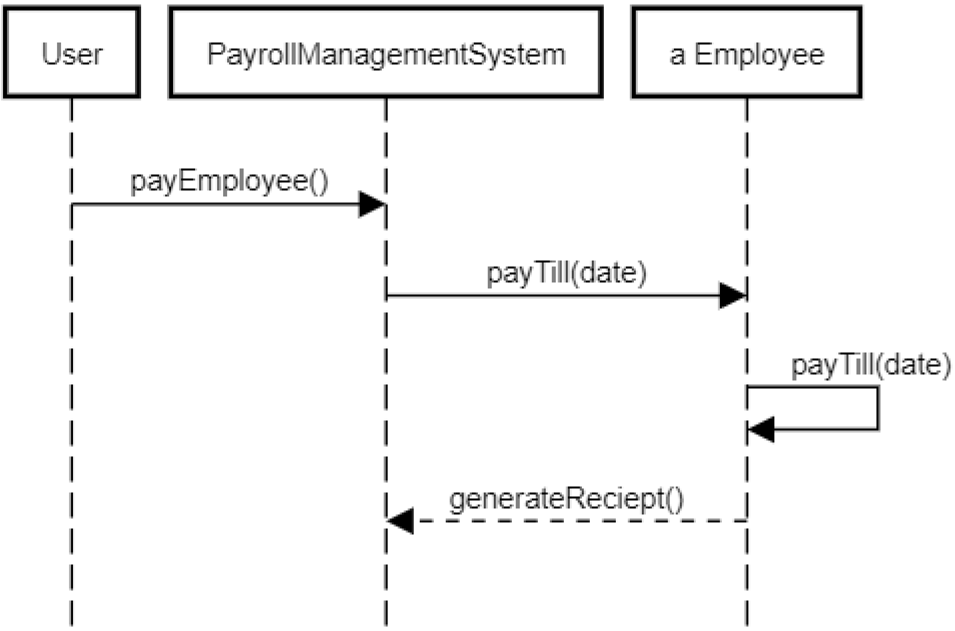
- Each class does a single well defined job with almost no side effects.
- Single Responsibility design has been followed.
- Further Requirments can be added easily and quickly with minimal changes using the testing system.

Sequence Diagrams

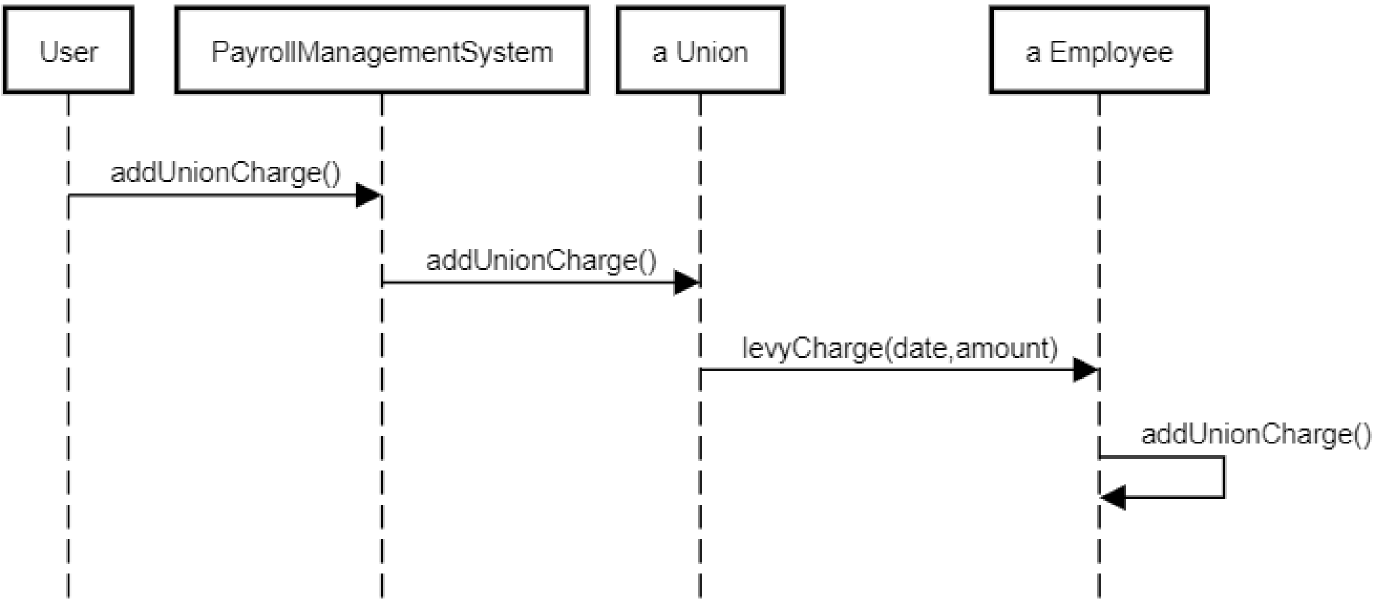
Adding/Removing Members from Union



Paying Employee Weekly/Monthly



Levying Union Charge


User
PayrollManagementSystem
a Union
a Employee
addUnionCharge()
addUnionCharge()
levyCharge(date,amount)
addUnionCharge()