Table of Contents

[System Overview 2](#_Toc69654404)

[Design Goals 2](#_Toc69654405)

[System Design 2](#_Toc69654406)

[Presentation Layer 3](#_Toc69654407)

[Employees 3](#_Toc69654408)

[HR/CEO/Site Manager Platform 3](#_Toc69654409)

[Finance Manager 4](#_Toc69654410)

[Application Layer 4](#_Toc69654411)

[System Modules 4](#_Toc69654412)

[System Structure 6](#_Toc69654413)

Dove Payroll System Design Document

# System Overview

Dove payroll is an electronic system that manages a company’s payroll. This it does by gathering employee information – which includes; basic information, next-of-kens, Benefits, and Salary Profiles – to calculate their taxes and net payable salaries. All the calculation for each of the employees are compiled on a paycheque which would be used by the system to generate the payroll for that period.

The proposed system manages the payroll for many companies in any of the configured tax regions. A tax region is any geographical region that has its basic information and tax definitions in the system’s data source. During tax calculations, the system would pull tax definitions of the tax region to which the company is registered.

Moreover, companies can use this system to efficiently manage employee records and salary disbursements. Employees can manage their basic information (on approval of the HR), make salary change requests to the company HR and view their paycheques. The HR can register and manage employee basic information, add and manage employee’s salary profiles and approve paycheques. The CEO can perform any action at the company level. The site manager can perform the functions of the HR related to employee basic information while the Finance officer is responsible for tax and salary calculations which creates the paycheques and hence the company payroll for the specified period.

Furthermore, the system can generate reports on different system entities including employee information, paycheques, salary profiles and payrolls which are accessible to Employees of with the appropriate access rights at the company level.

# Design Goals

The main goals of the system are as follows:

* Manage the Company payroll (includes paycheques, salary profiles, salary advances and company benefits)
* Manage company Employees (includes employee basic information, next-of-kin and benefits and insurances and/or pension)
* Manage company details (includes company information, tax region, PAYE definitions, employees and payroll)

# System Design

The architecture of the system is a Layered type and consists mainly of three (3) layers;

* Presentation
* Application
* Data Management.

The presentation layer holds the user interfaces for each of the system users which would be generating requests as they use the system. The subsystems in the Application layer receives requests to or from the presentation layer, process them and sends results back to the system users or data source depending on the nature of the request. The Data management layer as the name suggests, controls the flow of data to and from the data source.

A detailed description of the sub systems in each of these layers is as follows

## Presentation Layer

As mentioned earlier, this layer hosts the user interfaces for the different users interacting with the system. It consists of four (4) main components, namely:

* Employee
* HR/CEO
* Finance manager
* Site manager

### Employees

This is the base layout for all the users of the system. It consists of the following platforms:

* Registration
  + Basic information
  + Next-of-kin
* Salary Profile
  + Basic profile
  + Benefits
  + Salary Advances
* Paycheques
* Dashboard

From the Registration platform, the employee can perform management actions such as; updating their temp profile and request for profile updates to the company HR. The Salary profile platform consists of three (3) sub platforms and provides the view and update request functionality to the employee for any of the sub components in the salary platform. From the paycheque platform however, the employees can view their paycheques and monitor its approval status. The dashboard provides a summary of the employee profile and activities.

### HR/CEO Platform

This is one of the main platforms on the system as it has access to most of the system entities at the company level. It consists of the employee base functions and the following:

* Company
  + Profile
  + Benefits
  + Subscription
* Employee
* Salary profile
  + Employee salary profiles
  + Employee Benefits
  + Salary Advances
* Payroll
  + Paycheques
  + Payroll

Unlike a regular employee, the HR can actually update their basic profiles, salary profiles and can approve or unapproved their paycheques. Furthermore, from the company page, the HR can manage company details including; reading, updating or deleting any of the company sub components. These sub components include; company profile, company benefits and subscriptions.

Moreover, from the employee platform, the HR can manage employee profiles including their next-of-kens. This include add, update, delete or suspend an employee. The salary profile platform allows the HR to add, update and remove employee salary profile. It also includes managing sub components of the salary profiles which includes; employee benefits and salary advances. The payroll platform handles paycheques reviews and/or approvals and generating the payroll. In the payroll platform, the HR can view a list of payrolls generated timestamps of the company with an additional functionality of generating the payroll for a selected timestamp.

### Site Manager Platform

This platform has access to the employee basic profiles and have management access to any of the employee records except for approvals. Apart from the employee base platform, they have the following additional platforms

* Employees
  + List of employees

### Finance Manager

The finance manager if responsible for calculating taxes and generating payrolls for all the employees in the company. Therefore, apart from the regular employee base platform, they have the following additional platforms:

* Employees
  + Salary Profiles
  + Salary Advances
* Payroll
  + Paycheques
  + Payroll

Like all employees, the Finance manager can only request for basic profile or salary change to the HR. However, the Finance manager can view the salary profiles and salary advances of other employees from the employee platform and can make salary change requests on behalf of the other employees to the HR. Nevertheless, they cannot directly make changes to the employee’s salary profiles.

From the payroll platform, the Finance manager can make tax and salary calculations and also generate employee paycheques and company payrolls for a particular time period. The sub-platforms would hold a list of time periods that the paycheques or payrolls were generated and provide read, update or delete functions for each of the time periods.

## Application Layer

This layer holds the system logic and is responsible for processing requests to or from the presentation layer. This layer holds the different classes in the system which is separated into modules. The modules are them made to work with a particular structure which would allow for a smooth flow of information between the Presentation and Data management layers. The follow paragraphs provide details about the system modules and structure.

### System Modules

The system is made up of different components, each of which forms a module. These modules include:

* Region
* Company
* Employees
* Salary

#### Region Module

This module is concerned with the location and tax definitions of a company in the system. The two (2) classes found in this module are Tax Region and PAYE. The Tax Region class specifies the location of the company and also links the company with a tax definition in the PAYE class. Therefore, every Tax region needs to have a PAYE definition and a PAYE should be used by one and only one region. This setup defines an Composition relationship between the two classes and can be seen in the class diagram in figure \*.

[[ class diagram for region module ]]

#### Company Module

In this module, all company related activities can be done. The Activities includes company registration, subscriptions, and company benefits management. It includes the Company, Subscription, Payment methods and Company Benefit classes. During registration, a company profile is submitted to the system and a subscription type chosen before payment is made with any of the system available payment methods. After company registration is completed, the company benefits can be defined by one of the company level users (e.g., CEO, HR).

Nevertheless, a Company can only choose a single Subscription type but a subscription type can be used by many companies. Moreover, a company can have many configured benefits and those benefits can only be used by that company. These relationships and the corresponding attributes and methods can be seen in the class diagram in figure \*.

[[ class diagram for the company module ]]

#### Employees Module

This module is responsible for all employee related activities. These includes; basic profile and next of kin profile updates. After a company is created, a company level user would register employees by adding their basic profiles and sending them a registration invitation. Upon receiving this information, the employees would be given a temporal login on the platform where they can update their basic and next of kins profiles and send to the HR for approval. After approval from the HR or any other company level user, they would have an account on the system that is associated with the related company. There are only two classes in this module; employee and next of kin. However, there are different employee positions and each of these positions have their predefined roles on the system at the company level. Therefore, an employee can either be the CEO, HR, Site Manager, Finance manager, Auditor or any other employee. With this, the employee class is defined as an abstract class with each of the associated positions as its concrete sub classes. This would allow the system to have and employee type that can perform both generic and specific operations at the company level. Moreover, an Employee can only have one next of kin which can in turn belong to only one employee profile. This makes the relationship between the Employee and the associated Next of kin that of composition. The class structure and the relationship between the classes in this module is shown in figure \*.

[[ class diagram for the company module ]]

#### Salary Module

The salary module is responsible for all salary related operation. After the employee profiles are created and approved, the HR or related company level user would have to update the salary profiles of each employee. Each employee would also be given a list of company configured benefits (see company module) which would be used together with the salary profile to calculate the employee’s net salary. Moreover, as mentioned in the company module, each company would be linked to a region and thus a PAYE definition. The associated PAYE would be used together with the calculated net salary and employee type to calculate the employee taxes. The tax calculations, net salary and salary advances would be compiled on a Paycheque which would show the net payable salary of the employee. However, the paycheques for each employee generated during a particular period would be used to generate the company payroll for that same period. This introduces a new class in this module called Salary Period, in addition to the Employee Benefits, Salary Profiles and Paycheques classes.

A salary profile can have many benefits associated with it and that same benefits can be associated with many other employees. Figure \* below shows the different classes in this module and their relationships with each other.

[[ class diagram for salary module ]]

### System Structure

The system uses the Model, View, Controller and Language (MVCL) structure to carry out all its operations. The Models would provide access to the data securely stored in the database while the Controller would be in charge of processing the request to and from the View. The Language however, would be in charge of rendering the texts onto the View and it would do so according to the selected language. However, the general classes would form types for the entire system and would be used by the models and the controllers to process requests in the system. The structure would have their module definitions as shown below.

General classes / Payroll Types

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name | Attributes | Constructors | Methods |
| Region | | | |
| TaxRegion | Private int regionID  Private PAYE paye  Private int pensionRate  Private int medicalInsuranceRate  Private Company<> <companies>  Private String regionName | TaxRegion()  TaxRegion(regionID)  TaxRegion(private fields other than regionID) | Public setters and getters () |
| PAYE | Private String payeID  Private String upperScheme  Private String middleScheme  Private String lowerScheme | PAYE()  PAYE(payeID)  PAYE(private fields other than payeID) | Public setters and getters () |
| Company | | | |
| Company | Private (type as required) basicInfo  Private int companyId  Private Region region  Private SubscriptionType subscriptionType  Private Employees<> <employees>  Private CompanyBenefits<> <companyBenefits>  Private Subscription subscription | Company ()  Company (companyID)  Company (company basic info private fields) | Public setters and getters () |
| SubscriptionType | Private int typeID  Private int subscriptionType  Private int cost  Private Subscription<> <susbcriptions>  Private Company<> <companies> | SubscriptionType()  SubscriptionType (typeID)  SubscriptionType (private fields other than typeID) | Public setters and getters () |
| Subscription | Private int subscriptionID  Private SubscriptionType subscriptionType  Private int status  Private date timeRemaining, date,Created, lastUpdated | Subscription ()  Subscription(subscriptionID)  Subscription (private fields other than subscriptionID) | Public setters and getters () |
| PaymentMethod | Private methodID: int  Private methodName: String  Private otherInfo | PaymentMethod()  PaymentMethod(methodID)  PaymentMethod(other private fields) | Public setters and getters() |
| CompanyBenefit | Private int benefitID  Private String benefitName  Private int benefitType  Private int amount  Private int companyId  Private Employee<> <employees> | CompanyBenefit ()  CompanyBenefit (benefitID)  CompanyBenefit (private fields other than benefitID) | Public setters and getters () |
| Employees | | | |
| Employee | Private int employeeID  Private int position  Private (as required) basicInfo  Private Company company  Private NextOfKin nextOfKin  Private SalaryProfile salaryProfile | Employee ()  Employee (employeeID)  Employee (employee basic info private fields) | Public setters and getters () |
| NextOfKin | Private int kinID  Private int employeeID  Private (as required) basicInfo | NextOfKin ()  NextOfKin (kinID)  NextOfKin(basic infor private fields) | Public setters and getters () |
| Salary | | | |
| SalaryProfile | Private int salaryProfileID  Private Employee employee  Private int salaryStructure  Private (as required) salaryDetails | SalaryProfile ()  SalaryProfile (salaryProfileID)  SalaryProfile (basic info private fields) | Public setters and getters () |
| EmployeeBenefit | Private benefitID  Private int employeeID  Private CompanyBenefit companyBenefit  Private date dateCreated, lastUpdated | EmployeeBenefit ()  EmployeeBenefit (benefitID)  EmployeeBenefit (private fields other than benefitID) | Public setters and getters () |
| PayCheque | Private int chequeID  Private Employee employee  Private PayPeriod payPeriod  Private int salaryCalcRslt  Private date dateCreated, lastUpdated | PayCheque ()  PayCheque (chequeID)  PayCheque (other private fields) | Public setters and getters () |
| SalaryAdvance | Private int advanceID  Private int employeeID  Private double amount  Private int status  Private date dateCreated, lastUpdated | SalaryAdvance()  SalaryAdvance(advanceID)  SalaryAdvance(private fields other than advanceID) | Public setters and getters () |
| PayPeriod | Private int periodID  Private String name  Private date dateFrom, dateTo, dateCreated | PayPeriod()  payPeriod(periodID)  PayPeriod(other private fields) | Public setters and getters () |

Note

* For the basic info attribute of the employee, see analysis document for the list of employee basic information and set the attribute type accordingly
* The position attribute specifies the position of the employee in the company. It is given as an integer value that is between 0 and 5. These values represents the following:
  + 0 – CEO
  + 1 – HR
  + 2 – Site Manager
  + 3 – Finance Manager
  + 4 – Auditor
  + 5 – Other Employees
* Notice the omitted attributes in the third constructor. The employee has all these attributes but are not created in the employee class or module. Creation of those attributes is solely the responsibility of other classes either in the same module or in a different one altogether. Nevertheless, these attributes should be updated just after addition of a new employee, or when creating the constructure with the employee ID. Furthermore, the get methods for these attributes should use the methods in their respective parent classes to add or fetch the attribute from the data source. With this setup, the employee class would always have all the attributes it needs to carry out all its operations.
* The employee has all the other attributes needed for the salary calculation an could be accessed form the salary attribute.
* The other private fields for the second constructure includes the employee, salaryDetails and salaryStructure)
* The required datatype for the salaryDetails attributes should be extracted from the analysis document.

System models

The model interacts with the database. To connect with the database, the DBConnect class is used and it would be extended by the other models to carry out their functions. The DBConnect class is defined as thus below.

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name | Attributes | Constructors | Methods |
| DBConnect | Private String dbName  Private String dbUsername  Private String dbPassword  Private String dbHost  Private int dbPort | DBConnect()  DBConnect(private fields) | Private setters and getters ()  Private connect (): Connection |

#### Region Module models

Classes in the region module models controls database information related to the company region and associated tax definitions. There are two main classes in the module:

* Model\_tax\_region
* Model\_PAYE

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name | Attributes | Constructors | Methods |
| TaxRegion\_Model extends DBConnect |  |  | Private addTaxRegion(taxRegion : TaxRegion): int[] (regionId, addStatus)  Private updateTaxRegion(taxregion : TaxRegion): int updateStatus  Private deleteTaxRegion(regionID: int): int deleteStatus  Private getTaxRegion(int regionId): TaxRegion  Public Static getAllTaxregions(): ArrayList<TaxRegion> |
| PAYE\_Model extends DBConnect |  |  | Private addPaye(paye : PAYE): int[] (payeID, addStatus)  Private updatePaye(paye: PAYE):int updateStatus  Private deletePaye(payeID : int): int deleteStatus  Private getPaye(payeID: PAYE): PAYE  Public Static getAllPaye(): ArrayList <PAYE> |

Note

* The TaxRegion and PAYE object used for the add and update methods should be initialized with the private data fields
* For the delete and get methods, the respective class can be called and its id extracted (e.g., TaxRegion taxRegion.regionID) to process the request.

#### Company Module Models

This module comprises of five(5) main classes;

* Company
* Subscription Types
* Subscriptions
* PaymentMethod
* Company Benefits

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name | Attributes | Constructors | Methods |
| Company\_Model extends DBConnect |  |  | Private addCompany(company: Company): int[] (companyId, addStatus)  Private updateCompany(company: Company): int updateStatus  Private deleteCompany(companyID: int): int deleteStats  Private getCompany(companyID: int): Company company  Public static getAllCompanies(): ArrayList<Conpany> |
| SubscriptionType\_Model extends DBComnnect |  |  | Private addSubscriptionTypes(subtype: SubscriptionType): int[] (subscriptionTypeID, addStatus)  Private updateSubscriptionType (subtype: SubscriptionType ): int updateStatus  Private deleteSubscriptionType(typeID: int): int deleteStatus  Private getSubscriptionType(typeID: int): SubscriptionType subscriptionType  Public statis getAllSubscriptions(): ArrayList<SubscriptionType> |
| Subscription­\_Model extends DBConnect |  |  | Private addSubscription(subscribe: Subscription): int[] (subscriptionId, addStatus)  Private updateSubscription(subscribe: Subscription): int updateStats  Private deleteSubscription(subscribeID: int): int deleteStats  Private getSubscription(subscribeID: int): Subscription subscription  Public static getAllSubscriptions(): ArrayList<Subscription> |
| PaymentMethod\_Model extends DBConnect |  |  | Private addPaymentMethod (method: PaymentMethod): int[] (methodID, addStatus)  Private updatePaymentMethod(method: PaymentMethod): int updateStats  Private deletePaymentMethod(methodID: int): int deleteStats  Private getPaymentMethod(emthodID: int): PaymentMethod method  Public static getAllPaymentMethods(): ArrayList<PaymentMethod> |
| CompanyBenefit\_Model  extends DBConnect |  |  | Private addCompanyBenefit(cbenefit: CompanyBenefit): int[] (benefitID, addStatus)  Private updateCompanyBenefit(cbenefit: CompanyBenefit): int updateStatus  Private deleteCompanyBenefit(benefitID: int): int deleteStatus  Private getCompanyBenefit(benefitID: int): CompanyBenefit companyBenefit  Public static getAllCompanyBenefits(): ArrayList<CompanyBenefit> |

#### Employee Module Models

The module consists of the following classes

* Employee
* Next-of-kin

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name | Attributes | Constructors | Methods |
| Employee\_Model extends DBConnect |  |  | Private addEmployee(employee: Employee): int[] (employeeID, addStatus)  Private updateEmployee(employee: Employee): int updateStatus  Private deleteEmployee(employeeID: int): int deleteStatus  Private getEmployee(employeeID: int): Employee employee  Public static getAllEmployees(): ArrayList<Employee> |
| NextOfKin\_Model extends DBConnect |  |  | Private addNextOfKin(kin: NextOfKin): int[] (nextOfKinID, addStatus)  Private updateNextOfKin(kin: NextOfKin): int updateStatus  Private deleteNextOfKin(kinID: int): int deleteStatus  Private getNextOfKin(kinID: int): NextOfKin nextOfKin  Public static getAllNextOfKins(): ArrayList<NextOfKin> |

Note

* The getEmployee and getAllEmployees methods fetches the employee profile as well as all the other attributes of the employee (including; next-of-kin profile, Salary profile and paycheques.
* Deleting an employee would also delete all its associated attributes including next-of-kin, Salary profile and Paycheques\*

Salary Module Models

This module consists of five (5) classes:

* Salary Profile
* Employee Benefits
* Paycheques
* PayPeriod
* Salary Advances

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name | Attributes | Constructors | Methods |
| SalaryProfile\_Model extends DBConnect |  |  | Private addSalaryProfile(salary: SalaryProfile): int[] (salaryProfileID, addStatus)  Private updateSalaryProfile(salary: SalaryProfile): int updateStatus  Private deleteSalaryProfile(salaryID: int): int deleteStatus  Private getSalaryProfile(salaryID: int): SalaryProfile salaryProfile  Public static getAllSalaryProfiles(): ArrayList<SalaryProfile> |
| EmployeeBenefit\_Model extends DBConnect |  |  | Private addEmployeeBenefit(benefit: EmployeeBenefit): int[] (benefitID, addStatus)  Private updateEmployeeBenefit(benefit: EmployeeBenefit): int updateStatus  Private deleteEmpoyeeBenefit(benefitID: int): int deleteStatus  Private getEmployeeBenefit(benefitID: int): EmployeeBenefit emoloyeeBenefit  Public static getAllEmployeeBenefits(): ArrayList<EmployeeBenefit> |
| Paycheque extends DBConnect |  |  | Private addPaycheque(cheque: PayCheque): int[] (chequeID, addStatus)  Private updatePaycheque(cheque: PayCheque): int updateStatus  Private deletePaycheque(chequeID: int): int deleteStatus  Private getPaycheque(chequeID: int): Paycheque paycheque  Public static getAllPaycheques(): ArrayList<Paycheque> |
| SalaryAdvance\_Model extends DBConnect |  |  | Private addSalaryAdvance(advance: SalaryAdvance): int[] (advanceID, addStatus)  Private upateSalaryAdvance (advance: SalaryAdvance): int updateStatus  Private deleteSalaryAdvance(advanceID: int): int deleteStatus  Private getSalaryAdvance(advanceID: int): SalaryAdvance salaryAdvance  Public static getAllSalaryAdvances(): ArrayList<SalaryAdvance> |
| PayPeriod\_Model extends DBConnect |  |  | Private addPayPeriod(period: PayPeriod): int[] (periodide, addStatus)  Private updatePayPeriod(period: PayPeriod): int updateStatus  Private deletePayPeriod(periodide: int): int deleteStatus  Private getPayPeriod(periodide: int): Payperiod payPeriod  Public static getAllPayPeriods(): ArrayList<Payperiod> |

Note

* Deleting a pay period also deletes all paycheques created within that period. \*

\*needs restriction or further clarification

System Controllers

The constructors relate with the views and hence uses the models to process user requests and control user permissions. The controllers for the different modules are as follows

Region Module Controllers

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name | Attributes | Constructors | Methods |
| TaxRegion\_Controller extends TaxRegion\_Model |  |  | Public addTaxRegion(taxRegion: TaxRegion): int[] (regionID, addStatus)  Public updateTaxRegion(taxRegion: TaxRegion): int updateStatus  Public deleteTaxRegion(regionID: int): int deleteStatus  Public getTaxRegion(regionID : int): TaxRegion |
| PAYE\_Controller extends PAYE\_Model |  |  | Public addPaye(paye :PAYE): int[] (payeID, addStatus)  Public updatePaye(paye :PAYE):int updateStatus  Public deletePaye(payeID :int): int deleteStatus  Public getPaye(paye :int): PAYE |

Note

Company Module Controllers

This module comprises of four main classes;

* Company
* Subscription Types
* Subscriptions
* PaymentMethod
* Company Benefits

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name | Attributes | Constructors | Methods |
| Company\_Controller extends Company\_Model | SubscriptionType\_Controller subtype  CompanyBenefit\_Controller benefit  Subscription\_Controller subscribe  PaymentMethod payment | Constructor() | Public addCompany(company: Company): int[] (companyId, addStatus)  Public updateCompany(company: Company): int updateStatus  Public deleteCompany(companyID: int): int deleteStats  Public getCompany(companyID: int): Company company  Public subscribe (company: Company): int subscriptionStatus |
| SubscriptionType\_Controller extends SubscriptionType\_Model |  |  | Public addSubscriptionTypes(subtype: SubscriptionType): int[] (subscriptionTypeID, addStatus)  Public updateSubscriptionType(subtype: SubscriptionType): int updateStatus  Public deleteSubscriptionType(typeID: int): int deleteStatus  Public getSubscriptionType(typeID: int): SubscriptionType subscriptionType |
| Subscription\_Controller extends Subscription\_Model |  |  | Public addSubscription(subscribe: Subscription): int[] (subscriptionId, addStatus)  Public updateSubscription(subscribe: Subscription): int updateStats  Public deleteSubscription(subscribeID: int): int deleteStats  Public getSubscription(subscribeID: int): Subscription subscription |
| PaymentMethod\_Controller extends PaymentMethod\_Model |  |  | Public addPaymentMethod (method: PaymentMethod): int[] (methodID, addStatus)  Public updatePaymentMethod(method: PaymentMethod): int updateStats  Public deletePaymentMethod(methodID: int): int deleteStats  Public getPaymentMethod(emthodID: int): PaymentMethod method |
| CompanyBenefit\_Controller extends CompanyBenefit\_Model |  |  | Public addCompanyBenefit(benefit: CompanyBenefit): int[] (benefitID, addStatus)  Public updateCompanyBenefit(benefit: CompanyBenefit): int updateStatus  Public deleteCompanyBenefit(benefitID: int): int deleteStatus  Public getCompanyBenefit(benefitID: int): CompanyBenefit companyBenefit |
| PaymentMethod |  |  |  |

Note

* The Company\_Controller in the main class in this module and so must access the other controllers in this module. Therefore, the other controllers are declared as attributes of the company controller and initialized in its no argument.
* The subscription method in the company controller class uses the subscriptionType and payment Method attribute in the Company type to complete the subscription. The subscription cost is extracted from the subscription type while the payment methodology is extracted from the selected payment method. Upon completion of the subscription, an insertion should be made to the subscription table through the subscription attribute.
* The payment method class is only found in the controller of the Company module. \* *to be added later*

Employee Module Controllers

Includes two classes:

* Employees
* Next-of-Kin

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name | Attributes | Constructors | Methods |
| Abstract Employee\_Controller extends Employee\_Model | Private NextOfKin\_Controller kin\_ctrl | Employee\_Controller () | Public abstract addEmployee(employee: Employee): int[] (employeeID, addStatus)  Public abstract updateEmployee(employee: Employee): int updateStatus  Public abstract deleteEmployee(employeeID: int): int deleteStatus  Public abstract getEmployee(employeeID: int): Employee employee  Public login (employee: Employee): Employee:  Public makeProfileUpdateRequest(employee: Employee)  Public requestSalaryAdvance(advance: SalaryAdvance): int requestStatus |
| CEOEmployee\_Controller extends Employee\_Controller | Private Company\_Controller company\_ctrl  Private SalaryProfile\_Controller salary\_ctrl  Private EmployeeBenefit\_Controller benefit\_ctrl  Private SalaryAdvance\_Controller advance\_ctrl  Private Paycheque\_Controller paycheque\_ctrl  Private PayPeriod\_Controller payPeriod\_ctrl | CEO\_employee\_Controller() | Implements abstract methods accordingly  Public renewSubscription(company: Company): int renewStatus  Public approveEmployee (employee: Employee)  Public approvePayCheque(cheque: Paycheque)  Public approveSalaryProfile(salary: SalaryProfile)  Public attributes processing functions |
| HR extends Employee\_Controller | Private Company\_Controller company\_ctrl  Private SalaryProfile\_Controller salary\_ctrl  Private EmployeeBenefit\_controller benefit\_ctrl  Private SalaryAdvance\_Controller advance\_ctrl  Private Paycheque\_Controller paycheque\_ctrl  Private PayPeriod\_Controller payPeriod\_ctrl  Private NextOfKin\_Controller kin\_ctrl |  | Implements abstract methods accordingly  Public renewSubscription(company: Company): int renewStatus  Public approveEmployee (employee: Employee)  Public approvePayCheque(cheque: Paycheque)  Public approveSalaryProfile(salary: SalaryProfile)  Public attributes processing functions |
| SiteManager\_Controller extends Employee\_Controller |  |  | Implements abstract methods accordingly |
| FinanceManager extends Employee\_Controller | Private SalaryProfile\_Controller salary\_ctrl  Private EmployeeBenefit\_Controller benefit\_ctrl  Private Paycheque\_Controller paycheque\_ctrl  Private PayPeriod\_Controller payPeriod\_ctrl | FinanceManager\_Controller () | Implements abstract methods accordingly  Public makeSalaryUpdateRequest(salary: SalaryProfile): int requestStatus  Public generatePaycheque(cheche: Paycheque): PayCheque  Public generatePayroll(period: PayPeriod): ArrayList<PayCheque> |
| Auditor\_Controller extends Employee\_Controller | Private SalaryProfile\_Controller salary\_ctrl  Private EmployeeBenefit\_Controller benefit\_ctrl  Private Paycheque\_Controller paycheque\_ctrl  Private PayPeriod\_Controller payPeriod\_ctrl | Auditor\_Controller () | Implements abstract methods accordingly  Public view/get functions for the associated attributes (controllers) |
| OtherEmployees\_Controller extends Employee\_Controller |  | OtherEmployees\_Controller() | Implements abstract methods accordingly |
|  | | | |
| NextOfKin\_Controller extends NextOfKin\_Model | Private int | NextOfKin\_Controller () | Public addNextOfKin(kin: NextOfKin): int[] (nextOfKinID, addStatus)  Public updateNextOfKin(kin: NextOfKin): int updateStatus  Public deleteNextOfKin(kinID: int): int deleteStatus  Public getNextOfKin(kinID: int): NextOfKin nextOfKin |

Note

* The employees are the main objects on the system and their position in the company determines their access to the other components of the system. Therefore, the employee controllers are defined as an abstract class (i.e., a generic employee controller) with its concrete classes being the different employee positions at the company level.
* Each concrete class of the Employee\_controller abatract class defines the set of functions that it is allowed to perform.
* Tasks are performed by different controllers in different modules of the system. Therefore, to give access to the concrete classes in the Employee\_controllers, these classes would have to uses these specific controllers as attributes. This was, they can access the different methods of their controller attributes.
* The no argument controllers in the concrete classes would initialize all the class controller attributes
* With reference to “Public attributes processing functions”, every class with this method should implement all the methods in its associated attribute’s controller classes. This reference is used both in the CEO and HR controllers.
* With reference to “Public view/get functions for the associated attributes (controllers)”, every class with this method should only implement the view/get methods of its associated controller attributes.
* Referring to “Implements abstract methods accordingly”, the classes with this should implement the generic according to the functions in the system. Functions/role of each employee in the system can be found in the analysis document.
* Notice that the NextOfKin controller is also a generic attribute of the Employee controller. This means all the concrete classes of the Employee\_controller class have access to the NextOfKin controller methods.

Salary Module Controllers

This module consists of five (5) classes:

* Salary Profile
* Employee Benefits
* Paycheques
* PayPeriod
* Salary Advances

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name | Attributes | Constructors | Methods |
| SalaryProfile\_Controller extends SalaryProfile\_Model |  |  | Public addSalaryProfile(salary: SalaryProfile): int[] (salaryProfileID, addStatus)  Public updateSalaryProfile(salary: SalaryProfile): int updateStatus  Public deleteSalaryProfile(salaryID: int): int deleteStatus  Public getSalaryProfile(salaryID: int): SalaryProfile salaryProfile |
| EmployeeBenefit\_Controller extends EmployeeBenefit\_Model |  |  | Public addEmployeeBenefit(benefit: EmployeeBenefit): int[] (benefitID, addStatus)  Public updateEmployeeBenefit(benefit: EmployeeBenefit): int updateStatus  Public deleteEmpoyeeBenefit(benefitID: int): int deleteStatus  Public getEmployeeBenefit(benefitID: int): EmployeeBenefit emoloyeeBenefit |
| PayCheque\_Controller extends Paycheque\_Model |  |  | Public addPaycheque(cheque: PayCheque): int[] (chequeID, addStatus)  Public updatePaycheque(cheque: PayCheque): int updateStatus  Public deletePaycheque(chequeID: int): int deleteStatus  Public getPaycheque(chequeID: int): Paycheque paycheque |
| SalaryAdvance\_Controller extends SalaryAdvance\_Model |  |  | Public addSalaryAdvance(advance: SalaryAdvance): int[] (advanceID, addStatus)  Public upateSalaryAdvance (advance: SalaryAdvance): int updateStatus  Public deleteSalaryAdvance(advanceID: int): int deleteStatus  Public getSalaryAdvance(advanceID: int): SalaryAdvance salaryAdvance |
| PayPeriod\_Controller extends PayPeriod\_Model |  |  | Public addPayPeriod(period: PayPeriod): int[] (periodide, addStatus)  Public updatePayPeriod(period: PayPeriod): int updateStatus  Public deletePayPeriod(periodide: int): int deleteStatus  Public getPayPeriod(periodide: int): Payperiod payPeriod |