Employee Management System
Helper Document

Name:Shreyas Ketkar
Id:2020A7PS0075P

Description Of Classes

1. EmployeeTable-class to hold basic details common to both employee and manager along with their respective getter and setter functions.(has constructor present)

    - checkPassword()-function to check if the format of the password entered contains sufficient variable characters to make it secure.
    - checkEmail()-function to check if the email entered is of the company i.e with the domain "@company.in".
    - isValidDate()-function to check if the date is in DD/MM/YYYY format which returns a boolean value
    - checkNumber()-function to check if the phone number contains only digits and is of 10 digits.

2. Details(extends EmployeeTable)-class to take details of employee/manager
    - detailsInserter- function to take the input for the data members of employee and manager classes in a list and returns that list

3. Employee (extends EmployeeTable)-class that contains some more data members specific to an employee and respective getter and setter functions.

4. Manager (extends EmployeeTable)-class that contains some more data members specific to a manager and respective getter and setter functions.

5. IdValidator-class with methods to validate ID
    - **validateEmpId()**-boolean function to check if the format of id matches that of an employee
    - **validateMngId()**-boolean function to check if the format of id matches that of a manager.
6. EmployeeAdder()-class with methods to add a manager/employee

7. EmployeeUpdater-class with method to update details of employee/manager and remove an employee/manager.
   - **updateEmployee()**-function to update the details of employee/manager using generics.
   - **removeEmployee()**-function to remove an employee/a manager using generics.
8. DetailsViewer-class with methods to display the details of employee and manager
   - **viewEmpDetails()**-function to view the details of employee by using the Hashmap and the id of the employee
   - **viewMngDetails()**-function to view the details of manager by using the Hashmap and the id of the manager
9. EmployeeLeave-class with methods to apply for leave and check the leave status.
   - **applyForLeave()**-function for the employee to apply for a leave (which is granted only if total number of leaves taken by the employee are less than 10 days)
   - **leaveStatus()**-function to check the status of the applied leave

Code Walkthrough

- In the beginning of the code a menu comes which provides the user an option to access information related to
  - Employee
  - Manager
  - Department
- Apart from this the user is given an Additional option of exiting the employee Management System
- If the User selected Employee/Manager as an option a menu comes up which asks the user whether he wants to add an employee/Manager or not.Additionally an option is provided to go back to the main menu
- When the user selects the option to add an Employee/Manager, a menu comes up asking the details of the employee.Note:-**The email of the employee should have the company domain in the end.i.e should end with @company.com or @company.in**.After the user has entered the details, an unique Id is generated corresponding to each new Employee/Manager.And the Employee/Manager record is updated.The Employee/Manager is internally stored in a HashMap corresponding to its Id generated.
- When the user does not want to add an Employee/Manager, In case there are no existing Employees/Managers in the company a menu pops up showing that there are no existing employees in the company.Otherwise the user is provided to option information related to the employee or manager
- In case of an employee the user is provided an option to
  - View the Employee details
  - Update the existing employee details(Phone number,Address,Email)

    - Check the Employee Pay details(Which is calculated according to the performance of the employee and the employee designation).**Note the performance of the employee is evaluated on an parameter named performance index(1-10) which tells how the employee has performed in an specific year**
    - Apply for a leave
    - Check leave status(which is updated depending on the number of leaves the employee has earlier taken)
    - Promote an existing employee to manager
    - Remove an employee
    - An additional option to go to the previous menu is provided
- In case of a Manager the user is provided an option to
    - View the Manager details
    - Update the existing manager details(Phone number,Address,Email)
    - Check the manager Pay details(Which is calculated according to the performance of the employee and the employee designation).**Note the performance of the Manager is evaluated on an parameter named performance index(1-10) which tells how the employee has performed in an specific year**
    - Remove manager
    - Search an employee working under the manager(based on the common department of the employee and manager)
    - Display the projects which the manager along with his team is working on
    - An additional option to go to the previous menu
- After going to the previous menu the user can select the option to go to the Main menu
- At the Main menu apart from choosing employee or a manager the user has an option to select Department
- In case the user selects Department a menu comes up asking the user to particularly select an department from the following departments:
    - IT department
    - HR department
    - Sales and Marketing Department
- In each department the user is provided options to
    - View the department Budget
    - View all the employees working in the department
    - View the Department Projects
    - An additional option is provided to go to the previous menu
- The user can repeat this process and can finally exit the Employee Management System.

Limitations

1. Data which is taken input from the user is stored temporary.The data vanishes after the program runs completely
2. We have not provided an user interface due to which it becomes a little difficult for the user to use the program
3. Multiple Try Catch blocks and While loops have been used in the Main.So this might make the readability of the code a little bad.And may cause an issue while debugging any error in the code. The code in the main block is a little lengthy which might cause issues while testing and debugging
4. Concrete classes have been used instead of abstraction

5. Many cases of redundant code can be seen.The problem can be avoided by creating separated classes for every different functionality

Future Work Possible

1. To handle the issue of Temporary data, A database can be maintained using SQL so that the Employee data is saved and can be used in the future.
2. A Graphical user Interface can be provided to the program using the java swing library,Or any other software.This would make the program user friendly.
3. Instead of directly making changes in the classes or main.The classes can be made abstract and for each change a new derived class can be created to implement the change

Analysis Of Design Patterns

OOP Principles

1. Classes should be open for extension closed for modification

Any new functionality should be implemented by adding new classes attributes and methods instead of changing the current ones.Either the user should be allowed to access the original class through abstraction or to implement the functionality through new derived classes.Suppose in the class where we are calculating the salary based on the parameter Performance Index,We later decide to add a different parameter on the basis of which the salary is calculated say if employee is temporary or permanent.Then we have to change the code in the class so as the change in the salary is reflected properly.Now suppose in future We again have to change the parameter on which salary is calculated it won't be feasible to change the code every time . Instead we can make the class abstract and move the implementation of calculating the salary to the derived classes

2. Loose Coupling Between Objects That interact

In the code we have made two different classes for the manager and employee.All the functions of the employee and manager have been separately defined So if there is an error in any particular function of those it can easily be identified and debugged.

3. Abstraction Over Concrete Classes

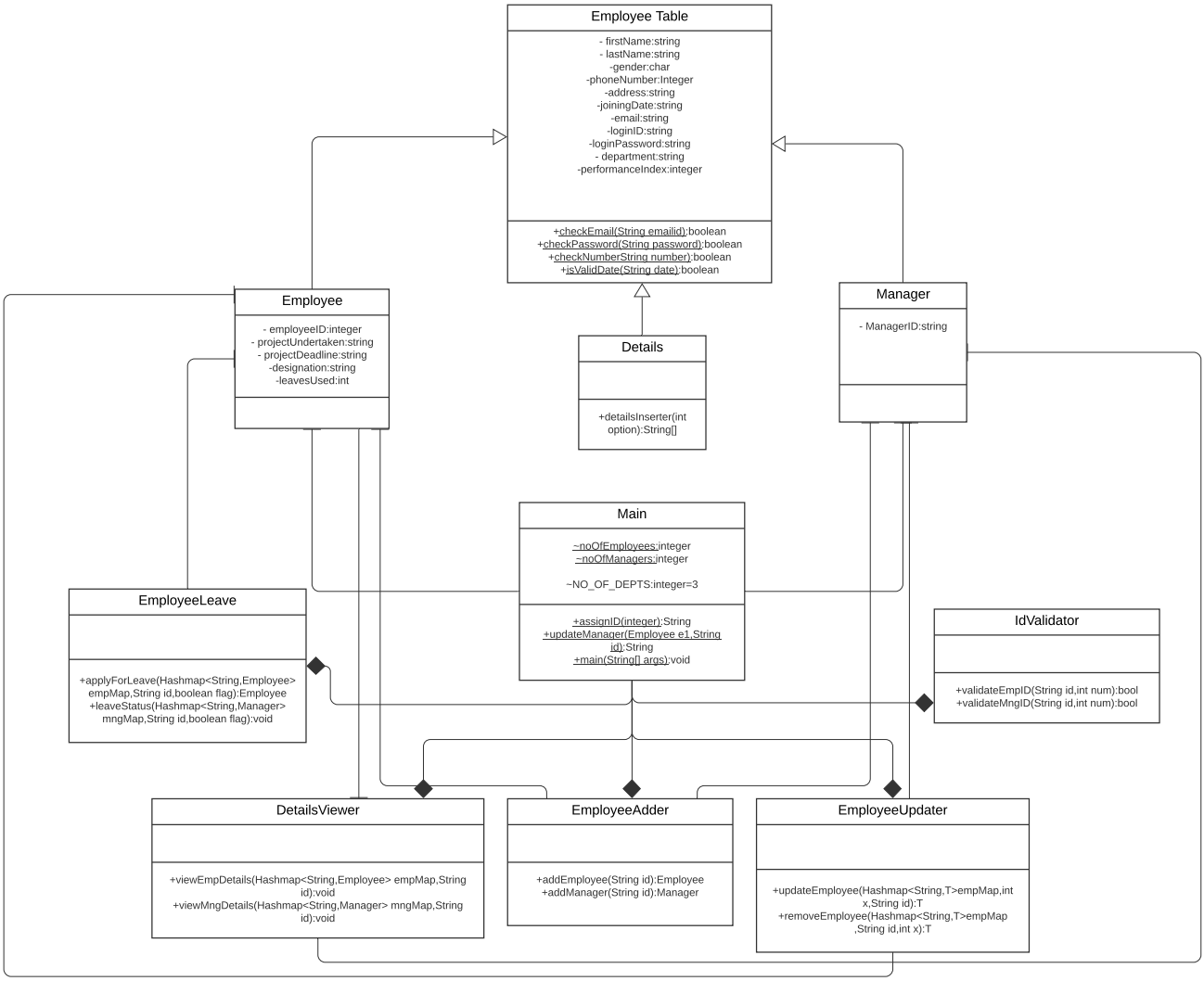
In this code abstraction has not been used rather concrete classes have been used.This may not cause a major issue when the number of users and the functionalities provided are less,However if the functionalities increase and we need the make any change in the code,The whole code would have to be rewritten.Using abstraction would help in better reusability of the code and would also help in easy testing

4. Encapsulate what varies

Although this principle has not been properly followed in the code,It can be used very well in an employee management System.Properties which might change are encapsulated so that they do not affect the properties which are going to remain fixed.In any employee management system there are certain properties such as the pay details of any employee/manager ,attendance of the employees ,projects assigned or the departments are constantly changing and should be encapsulated so that they do not affect the properties which remain fixed

Design Pattern

Although the following design pattern has not been used in the code but the best possible design pattern which can be used in an employee management system seems to be the **factory design pattern.**We create an abstract class for creating an object and let the subclasses decide which class to instantiate.In our case the employee Table class can be made an abstract class and the classes managers and employee can be instantiated as required.The advantage here is that the subclasses can decide what type of object is to be created.Also one of the fundamental principle of object oriented programming Loose coupling between objects that interact is followed.The code interacts with the abstract class and any other classes which extend that class.
Also the principle of Abstraction over Concrete Classes would be followed.The code can also be easily changed at run time instead of just compile time by allowing the subclasses to decide which classes to instantiate.


Employee Table
- firstName:string
- lastName:string
-gender:char
-phoneNumber:Integer
-address:string
-joiningDate:string
-email:string
-loginID:string
-loginPassword:string
- department:string
-performanceIndex:integer
+checkEmail(String emailid):boolean
+checkPassword(String password):boolean
+checkNumberString number):boolean
+isValidDate(String date):boolean
Employee
- employeeID:integer
- projectUndertaken:string
- projectDeadline:string
-designation:string
-leavesUsed:int
Details
+detailsInserter(int option):String[]
Manager
- ManagerID:string
Main
~noOfEmployees:integer
~noOfManagers:integer
~NO_OF_DEPTS:integer=3
+assignID(integer):String
+updateManager(Employee e1,String id):String
+main(String[] args):void
EmployeeLeave
+applyForLeave(Hashmap<String,Employee> empMap,String id,boolean flag):Employee
+leaveStatus(Hashmap<String,Manager> mngMap,String id,boolean flag):void
IdValidator
+validateEmpID(String id,int num):bool
+validateMngID(String id,int num):bool
DetailsViewer
+viewEmpDetails(Hashmap<String,Employee> empMap,String id):void
+viewMngDetails(Hashmap<String,Manager> mngMap,String id):void
EmployeeAdder
+addEmployee(String id):Employee
+addManager(String id):Manager
EmployeeUpdater
+updateEmployee(Hashmap<String,T>empMap,int x,String id):T
+removeEmployee(Hashmap<String,T>empMap,String id,int x):T