1. **Class Diagram with methods:**

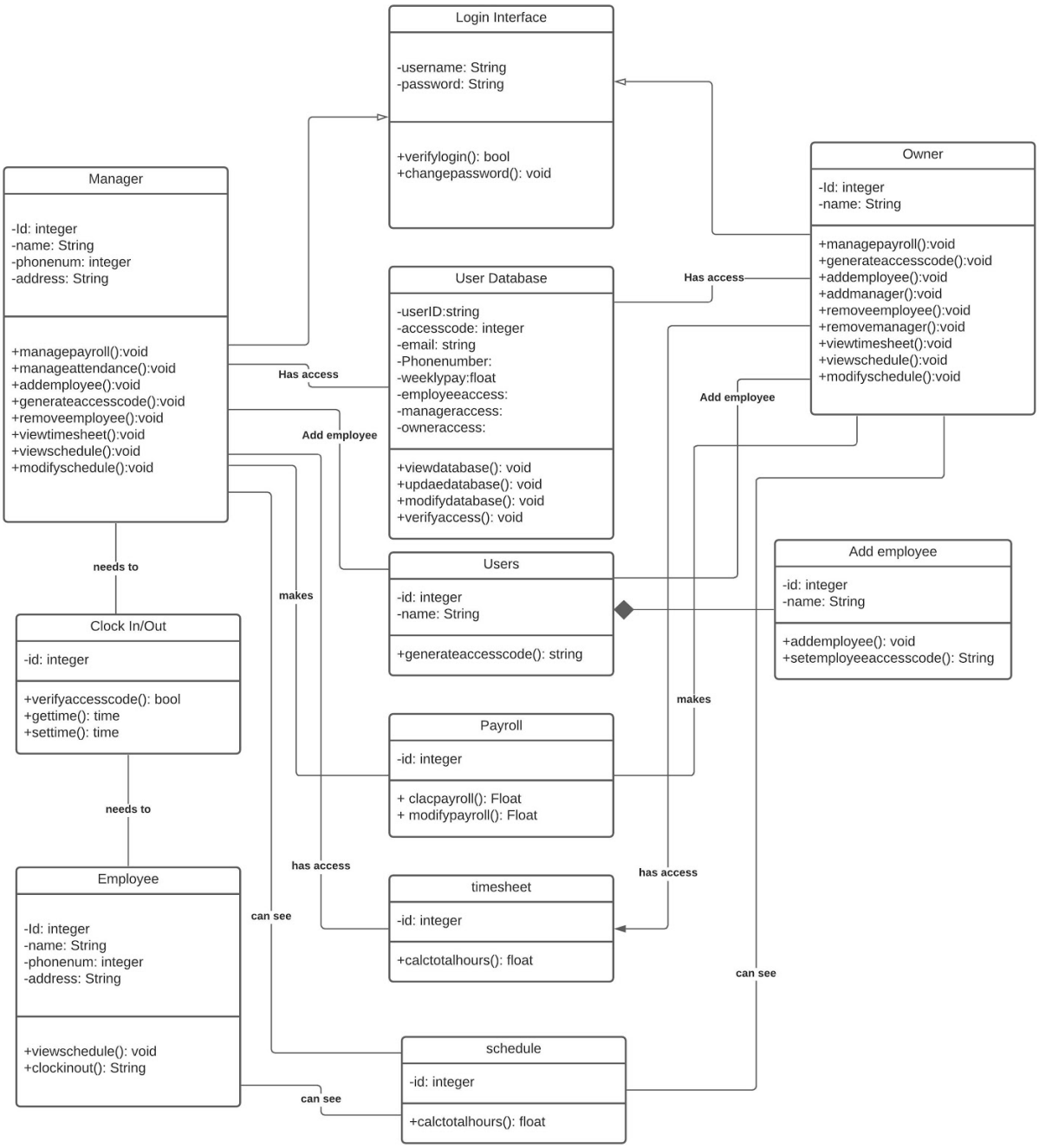

Fig: Class diagram for Employee Management

2. **Detailed Design**:

1. void managePayroll()
    if view_database = access_Type (ie. Manager)
    change/update payroll database info
    update database

2. void manageAttendance()
    if view_database = access_Type (ie. Manager)
    change/update employee attendance database info
    update database

3. void addEmployee()
    If employee identifiers do not match info in database
        employee identifiers are added to database
    else
        display “employee exists” error message

4. void generateAccessCode()
    stored_access_code = (Algorithm to computer access code from employee identifier info)

5. void removeEmployee()
    if employee identifies match database info
        employee identifiers are removed from database
    else
        display “employee not found” error message

6. void viewTimesheet()
    display Day of week, date, sign in and sign out for Sunday - Saturday work week.

7. void viewSchedule()
    display Day of week, date and scheduled hours for Sunday - Saturday work week.

8. void modifySchedule()
   prompt user to choose to modify the days of week, date or scheduled hours
   change schedules for the Sunday - Saturday work week
   update the database

9. boolean verifyAcessCode()
   If employee_access_code == stored_access_code
   return True;
   else
   return False;

10. void get_clockOut_Time()
    string clock_Out;
    clock_Out = system time

11. void get_clockIn_Time()
    string clock_In
    clock_In = system time

12. boolean verifyLogin()
    If User_ID == stored_ID && employee_password == stored_password
    return True;
    else
    return False;

13. void changePassword()
    if current_password == stored_password;
    { prompt user to enter new_password and new_password_match
    {while new_password != new_password_match;
    prompt user to enter new_password and new_password_match
    Else
    stored_password = new_password

```
        }
    Else
     display “Password does not match” error message
    allow user to attempt to re-enter password (x number of times)
    }
```

14. void viewDatabase()
    if view_database = access_Type (ie. Manager)
        return True;
    else
        return False;

15. void updateDatabase()
    if date_database = access_Type (ie. Manager)
        return True;
    else
        return False;

16. void modifyDatabase()
    if employee_name is found in nameList_array;
        prompt for changes to employee database variables
    else
    {   display “no match found” message;
        allow user to input info again if needed;
    }

17. void verifyAccess()
    if employee_name is found in nameList_array;
        prompt for changes to employee database variables
    else
    {   display “no match found” message;
        allow user to input info again if needed;
    }

18. float calcPayroll()
    float payRoll;
    get pay_rate value
    string total_hours= call calcTotalHours()
    payRoll = multiply pay_rate with total_hours
    return float payRoll

19. float modifyPayroll()
    float newPayroll;
    prompt user for employee_name
    if employee_name is found
    {
        prompt user to enter new float value for payroll
        store new float into newPayroll
        return and update database
    }
    else
    {
        display “no match found” message
        allow for new name input or exit with float newPayroll = 0.0
    }

20. float calcTotalHours()
    for x =0; x <= total_days; i++
    TotalsHours += daily_hours_array [x];

21. void addManager()
    string name, information;
    prompt user to enter information
    store info to string name and information
    call updateDatabase() and update with string info

```
22. void removeManager()
    prompt user to enter name
    if (manager exists in database)
    {
            call updateDatabase();
            delete manager from database
    }
    else
    display message telling user the manager does not exist

23. string setEmployeeAccessCode()
    if employee exists and accesscode is empty
    {
            generateAccessCode();
            map employee with generated access code
            return string employee and access code
    }
    else if employee exists and has an access code
    {
            prompt user if they wish to replace the access code
            if true
                    generateAccessCode();
                    map employee with generated access code
                    return string employee and access code
            else
                    return message that setting code is unnecessary
    }
    else
            prompt user that employee does not exist
```

3. Design Pattern (Adaptive pattern)

The Adapter Pattern applies the same idea to object-oriented programming by introducing an additional adapter class between an interface and an existing class. The adapter class implements the expected interface and keeps a reference to an object of the class you want to reuse. Adapter also provides all the advantages of information hiding without having to actually hide the implementation details. Our project will be easy to reuse if we use the adaptive pattern and most of the benefits and advantages of the adaptive diagram is common to our projects. The major reason to use the adaptive for this project is that all benefits of information hiding without having to actually hide the implementation details is available in this pattern and we needed it for the owner and manager class to hide some significant details from the employee and other users.