Milestone 1 Template

Requirements & Initial Class Diagram

Document Title: Milestone 1 – Requirements Analysis, Decomposition & Abstraction

1. Project Title: Human Resource Management System
2. Problem Description (5–7 sentences)

In this project,

- I aim to design a system: *manages employee information, attendance, salary calculations, and reports for a medium-sized company with various departments.*
- The main users will be HR officers who need to handle tasks like adding employees, recording attendance, and generating salary reports.
- The system will help solve the problem of inefficient manual HR processes by automating data storage, validation, and calculations based on business rules.

The key functions include: *managing employee details, tracking daily attendance with overtime, computing salaries with deductions and bonuses, and producing reports on attendance and pay.*

This project will apply OOP and Computational Thinking to decompose the problem and build an efficient solution. It ensures unique employee IDs, valid attendance records, and accurate salary computations while enforcing rules like unique dates per attendance and thresholds for low attendance.

3. Functional Requirements

(User Stories)

# An HR Officer,

I want to add a new employee with details like: *ID, name, department, job title, joining date, basic salary, and type (full-time or part-time)* so that new staff can be integrated into the system.

I want to update employee information such as *department or job title* so that changes in employee status are accurately reflected.

# An Accountant Officer,

I want to calculate monthly salary for an employee based on basic salary, overtime pay, and absence deductions so that payroll is processed correctly according to rules.

I want to generate reports listing employees with low attendance (exceeding a threshold like 3 absent days) so that performance issues can be identified.

I want to list the highest-paid employees based on calculated monthly salaries so that compensation insights can be gained.

# An Manager Officer,

I need an app that's easy to use and has a nice interface.

I want to record daily attendance for an employee, including date, status (Present/Absent/Leave), and overtime hours so that daily records are maintained and validated.

4. System Decomposition (CT: Decomposition)

The system can be divided into the following modules:

1. Employee Management: Handles adding, updating, removing, viewing, and searching employees.
2. Attendance Management: Manages recording, updating, and viewing attendance records, including calculations for working days and absences.
3. Salary Management: Computes salaries based on attendance data and generates salary reports.
4. Reporting: Produces lists of low-attendance and highest-paid employees.

Each module has a clear responsibility in the system.

5. Identified Classes (CT: Abstraction

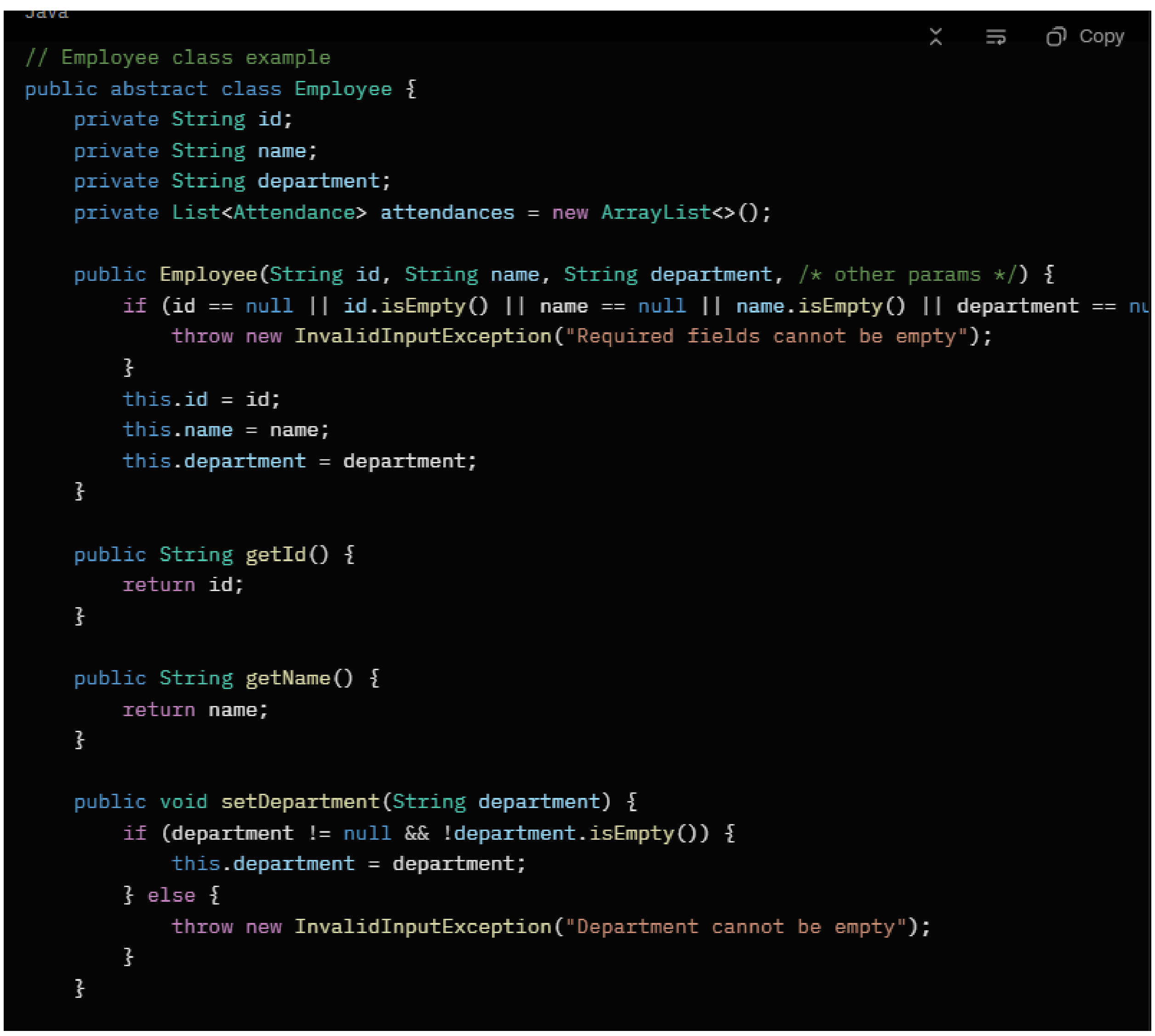
Class Name	**Attributes**	**Methods**
Employee (Abstract)	id (String), name (String), department (String), jobTitle (String), joinDate (Date), basicSalary (double), active (boolean), attendances (List<Attendance>)	Constructor with validation, getId(), getName(), setDepartment(String), setJobTitle(String), addAttendance(Attendance), updateAttendance(Date, String, int), getAttendances(), calculateWorkingDays(int month, int year), calculateAbsenceDays(int month, int year), calculateTotalOvertime(int month, int year), abstract calculateSalary(int month, int year), protected getOvertimeRate()

FullTimeEmployee	(inherits from Employee), overtimeRate = 80000.0 (double)	Constructor calling super(), override calculateSalary(int month, int year), override getOvertimeRate()
PartTimeEmployee	(inherits from Employee), overtimeRate = 50000.0 (double)	Constructor calling super(), override calculateSalary(int month, int year), override getOvertimeRate()
Attendance	date (Date), status (String), overtimeHours (int)	Constructor with validation, getDate(), getStatus(), setStatus(String), getOvertimeHours()
HRManagementSystem	employees (Map<String, Employee>)	Constructor, addEmployee(Employee), updateEmployee(String id, ...), removeEmployee(String id), getAllEmployees(), searchEmployees(String keyword), recordAttendance(String employeeId, Attendance), viewAttendanceHistory(String employeeId), calculateSalary(String employeeId, int month, int year), generateSalaryReport(int month, int year), listLowAttendance(int month, int year, int threshold), listHighestPaid(int month, int year), saveToFile(String), loadFromFile(String)

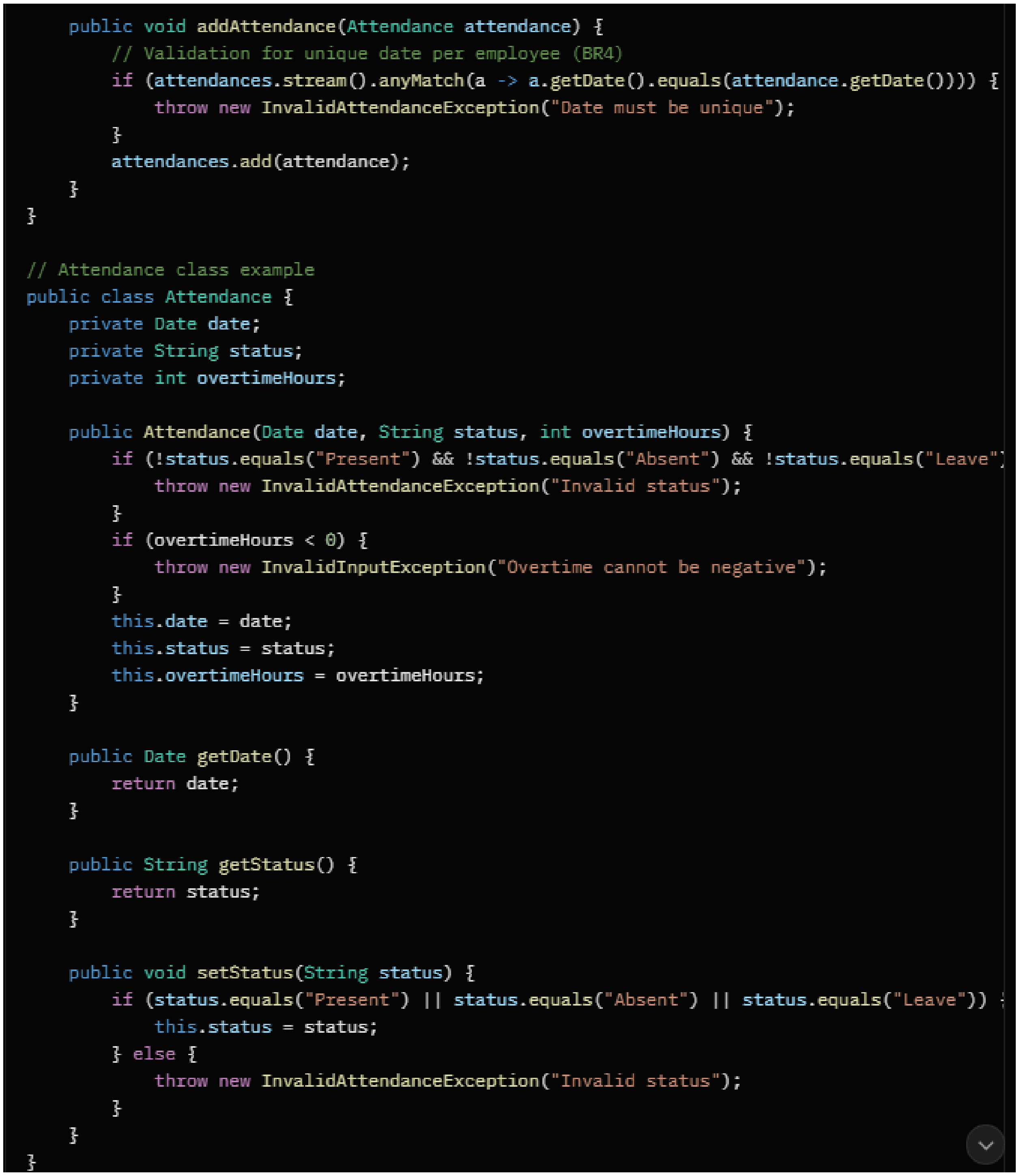
6. Relationships between real-world objects

The Employee class has a one-to-many Has-A relationship with Attendance (composition: one employee contains multiple attendance records, which exist only within the employee). HRManagementSystem has a Has-A relationship with Employee (aggregation: the system manages a collection of employees via a Map for quick lookup, but employees could exist independently). FullTimeEmployee and PartTimeEmployee have an Is-A relationship with Employee through inheritance, allowing polymorphism in salary calculations. No separate Salary class is needed as salary is dynamically computed from Attendance data. These relationships ensure single responsibility, with exceptions like EmployeeNotFoundException for handling errors in associations.

7. Encapsulation Evidence (Code Snippets)



```java
// Employee class example
public abstract class Employee {
    private String id;
    private String name;
    private String department;
    private List<Attendance> attendances = new ArrayList<>();

    public Employee(String id, String name, String department, /* other params */) {
        if (id == null || id.isEmpty() || name == null || name.isEmpty() || department == nu
            throw new InvalidInputException("Required fields cannot be empty");
        }
        this.id = id;
        this.name = name;
        this.department = department;
    }

    public String getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public void setDepartment(String department) {
        if (department != null && !department.isEmpty()) {
            this.department = department;
        } else {
            throw new InvalidInputException("Department cannot be empty");
        }
    }
```

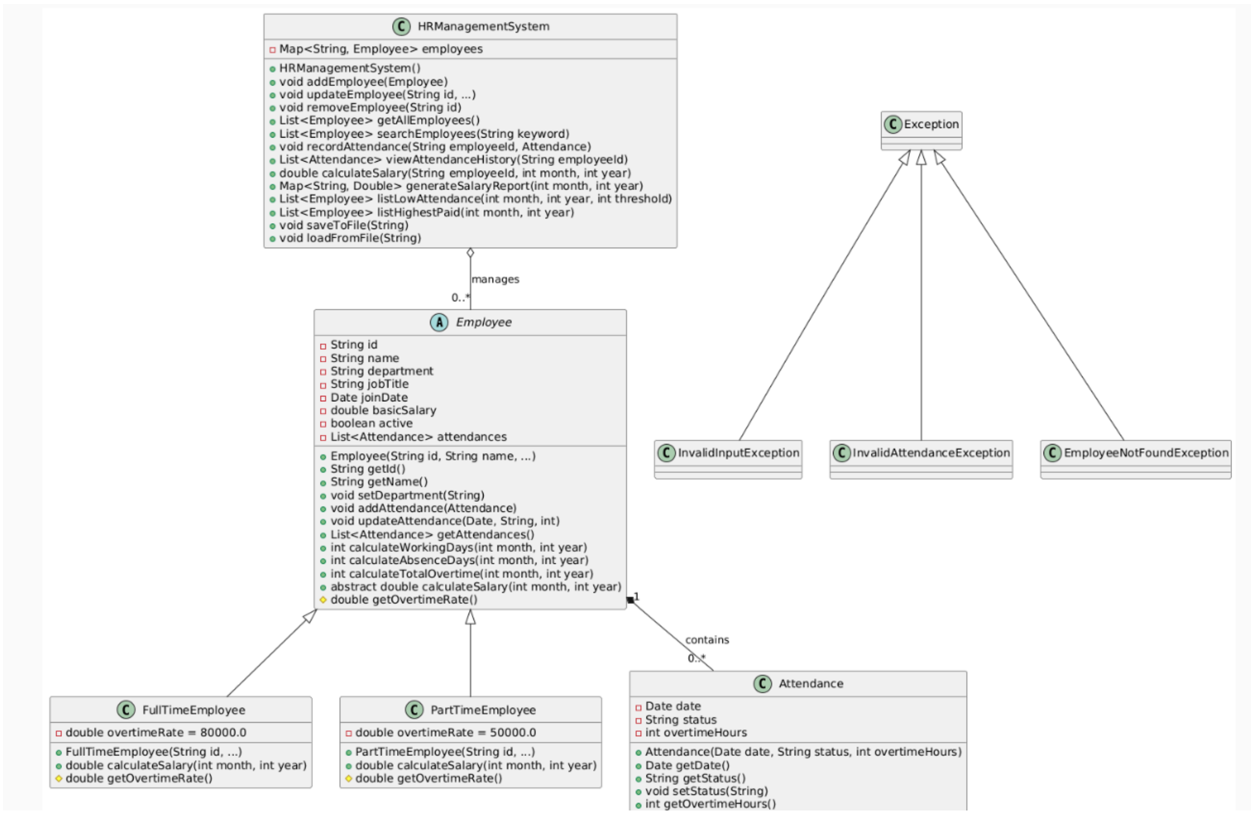


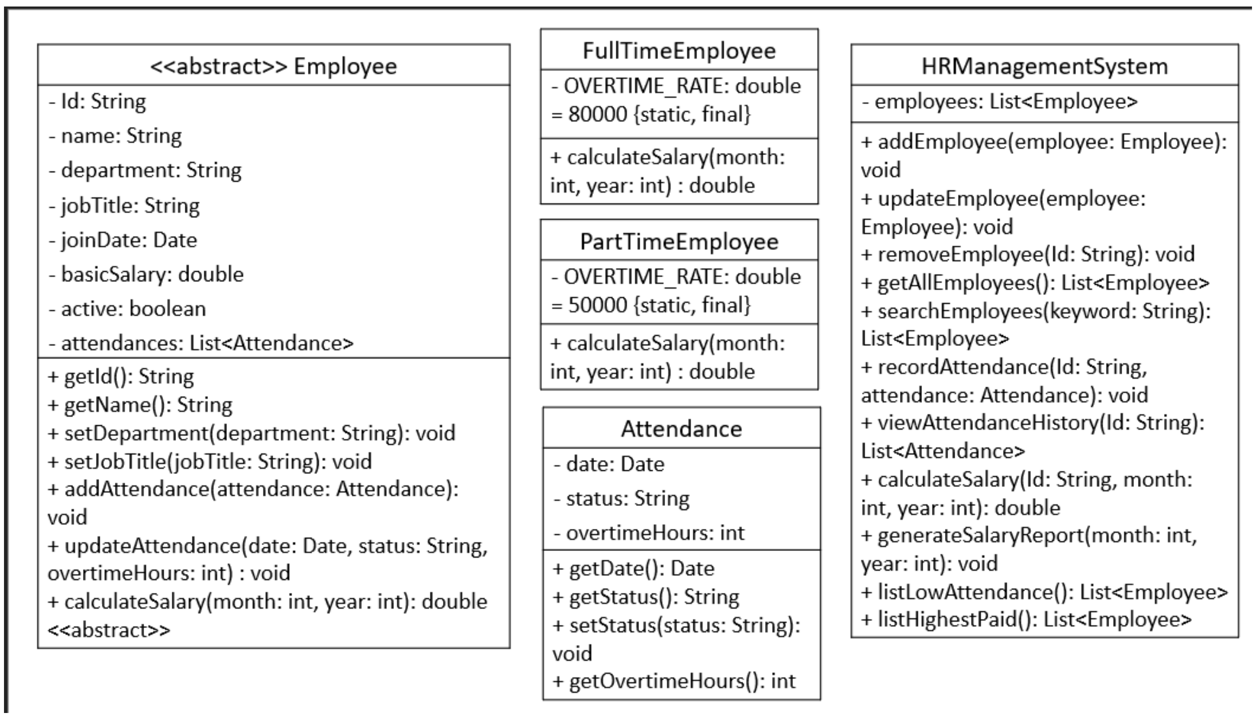
```java
    public void addAttendance(Attendance attendance) {
        // Validation for unique date per employee (BR4)
        if (attendances.stream().anyMatch(a -> a.getDate().equals(attendance.getDate()))) {
            throw new InvalidAttendanceException("Date must be unique");
        }
        attendances.add(attendance);
    }
}

// Attendance class example
public class Attendance {
    private Date date;
    private String status;
    private int overtimeHours;

    public Attendance(Date date, String status, int overtimeHours) {
        if (!status.equals("Present") && !status.equals("Absent") && !status.equals("Leave")
            throw new InvalidAttendanceException("Invalid status");
        }
        if (overtimeHours < 0) {
            throw new InvalidInputException("Overtime cannot be negative");
        }
        this.date = date;
        this.status = status;
        this.overtimeHours = overtimeHours;
    }

    public Date getDate() {
        return date;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        if (status.equals("Present") || status.equals("Absent") || status.equals("Leave"))
            this.status = status;
        } else {
            throw new InvalidAttendanceException("Invalid status");
        }
    }
}
```

8. Reflection (4–6 sentences)

In this milestone, I learned that decomposition helps me break down complex tasks into manageable parts, such as separating employee management from attendance and salary calculations. I also practiced abstraction by identifying the essential classes like Employee, Attendance, and HRManagementSystem, focusing on key attributes and methods while ignoring minor details. The relationships between classes, including Has-A for compositions like Employee containing Attendance, and Is-A for inheritance in employee types, model real-world hierarchies effectively. This analysis will guide the next milestone where I implement encapsulation and basic operations. Applying CT skills made the system design more structured, ensuring adherence to business rules and OOP principles. Overall, it highlighted how proper naming and relationships improve code maintainability.

Class diagram:


C HRManagementSystem
Map<String, Employee> employees
HRManagementSystem()
void addEmployee(Employee)
void updateEmployee(String id, ...)
void removeEmployee(String id)
List<Employee> getAllEmployees()
List<Employee> searchEmployees(String keyword)
void recordAttendance(String employeeId, Attendance)
List<Attendance> viewAttendanceHistory(String employeeId)
double calculateSalary(String employeeId, int month, int year)
Map<String, Double> generateSalaryReport(int month, int year)
List<Employee> listLowAttendance(int month, int year, int threshold)
List<Employee> listHighestPaid(int month, int year)
void saveToFile(String)
void loadFromFile(String)
manages
0..*
A Employee
String id
String name
String department
String jobTitle
Date joinDate
double basicSalary
boolean active
List<Attendance> attendances
Employee(String id, String name, ...)
String getId()
String getName()
void setDepartment(String)
void addAttendance(Attendance)
void updateAttendance(Date, String, int)
List<Attendance> getAttendances()
int calculateWorkingDays(int month, int year)
int calculateAbsenceDays(int month, int year)
int calculateTotalOvertime(int month, int year)
abstract double calculateSalary(int month, int year)
double getOvertimeRate()
1
contains
0..*
C Exception
C InvalidInputException
C InvalidAttendanceException
C EmployeeNotFoundException
C FullTimeEmployee
double overtimeRate = 80000.0
FullTimeEmployee(String id, ...)
double calculateSalary(int month, int year)
double getOvertimeRate()
C PartTimeEmployee
double overtimeRate = 50000.0
PartTimeEmployee(String id, ...)
double calculateSalary(int month, int year)
double getOvertimeRate()
C Attendance
Date date
String status
int overtimeHours
Attendance(Date date, String status, int overtimeHours)
Date getDate()
String getStatus()
void setStatus(String)
int getOvertimeHours()


<<abstract>> Employee
- Id: String
- name: String
- department: String
- jobTitle: String
- joinDate: Date
- basicSalary: double
- active: boolean
- attendances: List<Attendance>
+ getId(): String
+ getName(): String
+ setDepartment(department: String): void
+ setJobTitle(jobTitle: String): void
+ addAttendance(attendance: Attendance): void
+ updateAttendance(date: Date, status: String, overtimeHours: int) : void
+ calculateSalary(month: int, year: int): double <<abstract>>
FullTimeEmployee
- OVERTIME_RATE: double = 80000 {static, final}
+ calculateSalary(month: int, year: int) : double
PartTimeEmployee
- OVERTIME_RATE: double = 50000 {static, final}
+ calculateSalary(month: int, year: int) : double
Attendance
- date: Date
- status: String
- overtimeHours: int
+ getDate(): Date
+ getStatus(): String
+ setStatus(status: String): void
+ getOvertimeHours(): int
HRManagementSystem
- employees: List<Employee>
+ addEmployee(employee: Employee): void
+ updateEmployee(employee: Employee): void
+ removeEmployee(Id: String): void
+ getAllEmployees(): List<Employee>
+ searchEmployees(keyword: String): List<Employee>
+ recordAttendance(Id: String, attendance: Attendance): void
+ viewAttendanceHistory(Id: String): List<Attendance>
+ calculateSalary(Id: String, month: int, year: int): double
+ generateSalaryReport(month: int, year: int): void
+ listLowAttendance(): List<Employee>
+ listHighestPaid(): List<Employee>

Cấu trúc Project:

```
HRMS Project/
│
├── src/                                # 📦 Source code chính
│   ├── hrms/                  # 📦 Package chính của ứng dụng
│   │   ├── Main.java                           # 🚀 Chạy chương trình
│   │   │
│   │   ├── models/                            # 🧱 Các lớp đối tượng
│   │   │   ├── Employee.java                # 👥 Lớp trừu tượng nhân viên
│   │   │   ├── FullTimeEmployee.java        # 🕘 Nhân viên toàn thời gian
│   │   │   ├── PartTimeEmployee.java        # ⏳ Nhân viên bán thời gian
│   │   │   └── Attendance.java                 # 📅 Lớp chấm công
│   │   │
│   │   ├── services/                        # ⚙️ Xử lý nghiệp vụ
│   │   │   └── HRManagementSystem.java # 🧠 Logic chính hệ thống
│   │   │
│   │   ├── utils/                       # 🛠️ Tiện ích hỗ trợ
│   │   │   ├── FileHandler.java         # 💾 Đọc/ghi file
│    │    │    ├── Validator.java        # ✅ Kiểm tra dữ liệu
│    │    │    └── ConsoleUI.java           # 🖥️ Hiển thị menu
│    │    │
│    │    └── exceptions/               # ⚠️ Xử lý lỗi tùy chỉnh
│    │         ├── InvalidEmployeeException.java
│    │         └── AttendanceException.java
│    │
├── data/                               # 📁 Lưu dữ liệu (txt/dat files)
│    ├── employees.txt
│    └── attendance.txt
│
└── README.md                            # 📄 Hướng dẫn dự án
```