Hw-23SOLIDPrinciples

Single Responsibility Principle (SRP):

The EmployeeManager class has too many responsibilities. We could separate the persistence logic (save/load employees) into a separate class, for example EmployeePersistence.

```
public class EmployeeManager implements IEmployeeRepository  {
    private final String employeeFilePath = "employees.json";
    private Gson gsonInstance = new GsonBuilder().setPrettyPrinting().creat
    private final List<Employee> employeeList;
    public EmployeeManager() {
        gsonInstance = new GsonBuilder()
            .setPrettyPrinting()
            .registerTypeAdapter(Date.class, new DateAdapter())
            .create();
    List<Employee> loadedEmployees = loadEmployees();
    employeeList = (loadedEmployees != null) ? loadedEmployees : new ArrayL
    }

    public void addEmployee(Employee employee) {
        employeeList.add(employee);
    }

    public void updateEmployee(Employee updatedEmployee) {
    for (int i = 0; i < employeeList.size(); i++) {
        if (employeeList.get(i).getIdNumber().equals(updatedEmployee.getIdN
            employeeList.set(i, updatedEmployee);
            break;
        }
```

The Calculator class could be split into smaller, more specific classes, such as SalaryCalculator, DeductionCalculator, etc.

```
public class Calculator {
    public static double calculateTotalAmount(Employee emp) {
        double totalAmount = emp.getBasicSalary();
        totalAmount += emp.getBonuses();
        totalAmount += calculateOvertimeHours(emp.getOvertimeHours(), Consta
        totalAmount -= calculateTotalDeductions(emp);
        return Math.max(totalAmount, 0);
    }

    public static double calculateOvertimeHours(double hoursWorked, double r
        double overtimeHours = Math.max(hoursWorked - regularHours, 0);
        double overtimeValue = overtimeHours * hourlyRate * Constants.getOve
        return Math.round(overtimeValue * 100.0) / 100.0;
    }

    public static double calculateReserveFunds(double basicSalary) {
        return Math.round(basicSalary * Constants.getReserveFundsPercentage(
    }
```

Open/Closed Principle (OCP):

We could create an IEmployeeRepository interface that defines methods to add, update, and delete employees. Then implement this interface for different storage types (MongoDB, JSON, etc.).

```
public class EmployeeToMongo  {
    private static final String CONNECTION_STRING = "mongodb+srv://yasisale
    private static final String DATABASE_NAME = "RolePaymentSystem";
    private static final String COLLECTION_EMPLOYEES = "employees";
    private static final String COLLECTION_PASSWORD="password";
    private static final String COLLECTION_CALCULATOR="calculator";
    private static final String COLLECTION_AMOUNT="amount";
    private static final String COLLECTION_SALARY_UPDATES = "salary_updates
    private static final String COLLECTION_PDF = "pdf_reports";

    private MongoClient mongoClient;
    private MongoDatabase database;

    public EmployeeToMongo() {
        this.mongoClient = createMongoClient();
        this.database = mongoClient.getDatabase(DATABASE_NAME);
    }

    private static MongoClient createMongoClient() {
        ServerApi serverApi = ServerApi.builder().version(ServerApiVersion.

        MongoClientSettings settings = MongoClientSettings.builder()
                .applyConnectionString(new ConnectionString(CONNECTION_STRI
                .serverApi(serverApi)
                .build();
```

Second:

```java
private void saveEmployeeToDatabase(Employee employee, MongoDatabase database) {
    MongoCollection<Document> collection = database.getCollection(COLLECTION_EMPLOYEES
    Document employeeDocument = new Document("id", employee.getIdNumber())
            .append("nombre", employee.getName())
            .append("apellido", employee.getLastName())
            .append("fechaContratacion", employee.getHireDate());

    try {
        collection.insertOne(employeeDocument);
        System.out.println("Empleado guardado exitosamente!");
    } catch (MongoException e) {
        e.printStackTrace();
    }
}

private void savePaymentDetailsToDatabase(EmployeePaymentDetails paymentDetails, Mongo
    MongoCollection<Document> collection = database.getCollection(COLLECTION_CALCULATO
    Document paymentDetailsDocument = new Document("overtimePayment", paymentDetails.g
            .append("reserveFunds", paymentDetails.getReserveFunds())
            .append("totalIncome", paymentDetails.getTotalIncome())
            .append("iessContribution", paymentDetails.getIessContribution())
            .append("biweeklyAdvance", paymentDetails.getBiweeklyAdvance())
            .append("foodDeduction", paymentDetails.getFoodDeduction())
            .append("totalExpenses", paymentDetails.getTotalExpenses())
            .append("netPayment", paymentDetails.getNetPayment())
            .append("employerContribution", paymentDetails.getEmployerContribution())
            .append("totalEmployeeCost", paymentDetails.getTotalEmployeeCost());
```