Brief Report on Application Design and Object-Oriented Principles Applied

---

1. Application Design

1.1. Project Structure

The project structure consists of a bunch of packages, each carrying out a kind of different functionality. Namely,

- **`model`**: includes the major data structures - `Employee` and `Department`.
- **`service`**: this package includes the business logic in classes - `EmployeeService` and `DepartmentService`, which control the operations concerning employee and department entities.
- **`exception`**: this is the package that holds the classes of custom exceptions - `EmployeeNotFoundException` and `DepartmentNotFoundException` meant for special error conditions.
- **`util`**: utility functions - `InputValidator`, validate the user inputs that come for processing.
- **`main`**: This contains the application entry point, `EmployeeManagementSystem`, interacting with the user through a command-line interface.

1.2. Class Design

- **`Employee Class`**:
    - Defines the employee entity with an ID, name, designation, salary, and department ID.
    - Provides constructors, getters, and setters along with a toString method to access and display employee information in any form with ease.

- **`Department Class`**:
    - Represents a Department entity that includes, among other attributes, id, name, and description.
    - Department class has constructors, getters and setters, as well as a toString method to handle the department information.

- **`EmployeeService` and `DepartmentService`:**
    - Service classes encapsulating the respective business logic of handling employees and departments.
    - They include respective methods for adding, removing, updating, and retrieving entities; therefore, respecting separation of concerns, and enhancing reuse of code.

---

2. Object-Oriented Principles Applied

2.1. Encapsulation

- **Implementation :**- Encapsulation is achieved by declarin the attributes for the classes Employee and Department as private, and then providing public methods (getters and setters) to access and modify these attributes.

- **Benefit**:- It helps to hide the internal implementation details, prohibiting the direct access of data in view such that any changes to the internal representation do not have any impact on other parts of the application.

2.2. Abstraction

- **Implementation**:- Service classes (`EmployeeService`, `DepartmentService`) provide an abstract interface for employee and department management, respectively. The central application interacts with these services without any concern about the details of the storage of data or manipulation of data.

- **Benefit**:- Abstraction will simplify the interaction with the system. Users perform operations using a very simple interface while complex logics are hidden at the service layer.

2.3. Inheritance

- **Implementation**:- Custom exceptions (`EmployeeNotFoundException`, `DepartmentNotFoundException`) inherit from the RuntimeException class. This is an example of inheritance, whereby an existing class is extended in order to implement a specialized exception for the application.

- **Benefits**:- Inheritance allows code reuse from the base `RuntimeException` class by the custom-derived exceptions. It reduces duplication and maintains consistency.

2.4. Polymorphism

- **Implementation**:- Polymorphism is subtly applied through method overriding, such as in the `toString` methods of classes `Employee` and `Department`. Each class implements `toString`, providing a personalized string representation of the objects.

- **Benefit**:- Polymorphism allows flexibility in the code, enabling different implementations of a method to coexist and be called based on the object type, improving the system's adaptability to changes.