Industry-Oriented Java Assignments (With Implementation Clues)

Assignment 1: Employee Management System

Create an `Employee` class with private fields: `id`, `name`, `salary`, and `department`. Provide public getters and setters for these fields. Create a constructor for initialization. Create a `Manager` class that inherits from `Employee` and adds fields `teamSize` and `projectName`. Write a method in `Manager` to display full manager details including team and project info. Create a main method to instantiate both classes and display their data.

Assignment 2: Online Banking System

Define a `BankAccount` class with private fields: `accountNumber`, `balance`, `ownerName`. Use methods `deposit()` and `withdraw()` to modify the balance. Create `SavingsAccount` and `CurrentAccount` subclasses with specific behavior: `SavingsAccount` adds `interestRate` and calculates interest; `CurrentAccount` has `overdraftLimit`. Demonstrate the working by creating objects of both classes and performing transactions.

Assignment 3: E-Commerce Product Catalog

Create a base class `Product` with fields: `productId`, `name`, `price`, `description`. Use encapsulation to protect data. Create subclasses like `Electronics` (with `warrantyPeriod`), `Clothing` (with `size`, `fabric`), and `Groceries` (with `expiryDate`). Implement a method `displayProductDetails()` to print product info. Create and display objects in main method to simulate a product catalog.

Assignment 4: University Course Registration System

Design a `Person` class with fields: `name`, `email`, `age`. Derive `Student` and `Professor` classes. `Student` should have `studentId`, `courseList`; `Professor` should have `subjectTaught`, `employeeId`. Use encapsulation to manage all fields. Implement methods to register students to a course and list them. Simulate registration using a `Course` class.

Assignment 5: Vehicle Rental Application

Build a `Vehicle` class with fields: `registrationNumber`, `brand`, `rentalRate`. Create subclasses `Car` (with `seatingCapacity`), `Bike` (with `engineCapacity`), and `Truck` (with `loadCapacity`). Include a method `calculateRental(int days)` that multiplies daily rate by days. Show polymorphism by calling `calculateRental()` from parent class reference.

Assignment 6: Hospital Patient Records

Create a base class `Patient` with private fields: `patientId`, `name`, `age`. Subclasses `InPatient` adds `roomNumber`, `admissionDate`; `OutPatient` adds `visitDate`, `consultationFee`. Encapsulate fields and override `displayPatientInfo()` method in each subclass. Create and print records for both types of patients.

Assignment 7: Library Book Management System

Design a `Book` class with fields like `title`, `author`, `ISBN`, `price`. Subclasses `PrintedBook` (with `pages`, `coverType`), `EBook` (with `fileSize`, `format`), `AudioBook` (with `duration`, `narrator`) inherit from `Book`. Override `displayDetails()` method to include specific fields. Use arrays or lists to store multiple books and display details of each.

Assignment 8: Online Food Delivery System

Create a base class `MenuItem` with `name`, `price`, `type`. Derive `VegItem` and `NonVegItem`. Add attributes like `calories`, `spiceLevel`. Implement a method `displayItem()` for printing full menu details. Create a class `Order` which contains multiple `MenuItem` objects and computes total bill. Use loops and encapsulated data handling.

Assignment 9: Insurance Policy Management

Create a `Policy` class with `policyNumber`, `holderName`, `amount`. Create subclasses `LifeInsurance` (with `nominee`), `HealthInsurance` (with `hospitalCoverage`), `VehicleInsurance` (with `vehicleDetails`). Encapsulate all data and implement method `calculatePremium()` in each subclass differently. Show inheritance and overriding with polymorphic calls.

Assignment 10: Smart Home Automation System

Design a base class `Device` with `deviceId`, `deviceName`, and `status`. Create subclasses `SmartLight`, `SmartThermostat`, and `SmartLock`. Each subclass implements its own version of `performAction()` (e.g., light turns on, thermostat adjusts temperature). Demonstrate how calling `performAction()` from the base class reference results in specific actions.