**Case Study: Westminster Health Centre Management System – Object-Oriented Design and Implementation**

**Purpose of the Case Study**

This coursework develops a **fully object-oriented Java system** for managing staff and operations in a small private health centre.

Students will **design, document, and implement** the system entirely from scratch using their knowledge of OOP principles.

By the end, students should be capable of:

1. Translating system requirements into **OOP-based designs (classes, inheritance, interfaces, polymorphism)**.
2. Implementing the design using Java in **IntelliJ**.
3. Using **collections, GUIs, event handling, exceptions, and file handling** effectively.
4. Producing a **maintainable, testable, and extendable** system suitable for a real-world scenario.

**Background Information (Scenario)**

The **Westminster Health Centre** is a small clinic that employs several **Doctors**, **Receptionists**, and other staff members.

Currently, all staff information—such as names, roles, dates of birth, and contact numbers—is stored manually in paper files.

The centre wishes to **automate staff management** to achieve the following objectives:

1. **Record staff details** (Doctors, Receptionists, and future roles).
2. **Search, edit, and remove** staff records easily.
3. **View staff details** in both **textual (console)** and **graphical (GUI)** forms.
4. **Generate simple statistics**, such as the number of doctors vs receptionists.
5. **Store and retrieve data** automatically using **files**.
6. Later, optionally integrate a **GUI** for interaction, similar to the one described in the official lab-based practical.

The clinic’s operations are straightforward—each staff member has an ID, personal information, and role-specific attributes.

Doctors maintain consultation data, while receptionists handle scheduling and front-desk duties.

The management expects the software to provide an interface for adding new staff, listing existing staff, and visualising them in a user-friendly way.

**System Concept**

**Core Entities**

1. **StaffMember (abstract class)** – the base for all staff.
2. **Doctor** – subclass of StaffMember; adds attributes like licence number, specialisation, and consultations per week.
3. **Receptionist** – subclass of StaffMember; adds desk number and working hours.
4. **Manager / Controller Class** – handles the collection of staff members and provides menu-based interactions.
5. **GUI Layer (optional, later stage)** – displays staff details in a table with interactive buttons.
6. **File Handling Layer (final stage)** – reads and writes staff data to a text or CSV file.

**Functional Requirements**

Students must implement the following **functional features** in stages (Weeks 05–10):

**1. Core Staff Management**

* Add a new staff member (doctor or receptionist).
* Display all staff information in the console.
* Search staff by ID.
* Remove staff by ID.

**2. Data Validation**

* Validate phone numbers (digits only).
* Validate date of birth format (dd/MM/yyyy).
* Validate numeric input (hours, consultations).

**3. Statistics and Reporting**

* Calculate total number of staff per role.
* Display average consultations per doctor.

**4. GUI Functionality**

* Display staff list in a JTable.
* Add buttons for “Add Staff”, “Show Statistics”, “Save Data”.
* Show data in pop-up dialogs.

**5. File Handling**

* Save staff list to a text or CSV file.
* Load data automatically at program start.

**6. Error Handling**

* Use try/catch to manage invalid inputs.
* Throw custom exceptions for duplicate IDs or invalid data.

**Non-Functional Requirements**

| **Aspect** | **Description** |
| --- | --- |
| **Language** | Java (JDK 17 or later) |
| **IDE** | Latest IntelliJ |
| **OOP Principles** | Encapsulation, inheritance, abstraction, polymorphism |
| **Data Structures** | Use ArrayList<StaffMember> |
| **User Interface** | Console-based first, Swing GUI later |
| **Validation** | Strict input validation and meaningful error messages |
| **Persistence** | File-based (CSV or text) |
| **Testing** | Use JUnit for at least three tests |

**System Design Guidelines**

**1. Class Hierarchy (to be designed by students)**

+---------------------------+

| StaffMember | ← abstract

+---------------------------+

| - staffId : String |

| - name : String |

| - surname : String |

| - dob : LocalDate |

| - phone : String |

+---------------------------+

| + getFullName() : String |

| + toString() : String |

+---------------------------+

/\

/ \

/ \

+---------------+ +------------------+

| Doctor | | Receptionist |

+---------------+ +------------------+

| - licenceNo | | - deskNo |

| - speciality | | - hoursPerWeek |

| - consultsPW | | |

+---------------+ +------------------+

**2. Manager Class**

WestminsterHealthCentreManager (implements interface HealthCentreManager)

* Holds an ArrayList<StaffMember> for storage.
* Provides methods for:

void addStaffMember();

void printStaffList();

void deleteStaffMember();

void runMenu();

void runGUI();

**3. GUI Design**

* Use JFrame for the main window.
* Use JTable with a custom TableModel.
* Use JButton for interactions.
* Optional: Add a summary popup using JOptionPane.

**Recommended Development Stages (Aligned to Teaching Weeks)**

| **Week** | **Topic Focus** | **Expected Progress** |
| --- | --- | --- |
| **05** | Fundamentals, UML, inheritance, polymorphism, Abstract classes, interfaces, | Design and implement StaffMember, Doctor, Receptionist and Add HealthCentreManager |
| **06** | Collections | use ArrayList |
| **07** | GUI programming | Implement a basic Swing window and table |
| **08** | Event handling, JUnit | Add buttons, listeners, and tests |
| **09** | Exception handling | Create custom exceptions and validate inputs |
| **10** | File handling | Add save/load features with proper exception management |

**Assessment Alignment**

| **Learning Outcome** | **Expected Evidence** |
| --- | --- |
| **LO2** | OOP architecture (class diagrams, abstract classes, inheritance) |
| **LO3** | Implementation of collections, menu, and search/sort functionality |
| **LO4** | GUI with events, models, and listeners |
| **LO5** | Exception handling, file I/O, and JUnit tests |

**📑 Expected Skill Mastery by Week 10**

* OOP Design (LO2)
* Collections and Data Structures (LO3)
* GUI Programming with Swing (LO4)
* Exception Handling & File Persistence (LO5)