Project Documentation: Victoria University UTMS Prototype

1. Introduction

This document outlines the design choices and demonstrates how core Object-Oriented Programming (OOP) principles have been applied in the Java-based prototype of the University Transport Management System (UTMS). The prototype manages data for Drivers, Vehicles, Routes, and supports three user roles: Students, Lecturers, and Transport Officers.

---

2. Design Overview

- **Packages**
    - vu.utms.app – Main program class
    - vu.utms.model – domain classes (User, Driver, Vehicle, Route) and interfaces.
    - vu.utms.dao – Data-Access Objects for each entity (CRUD operations).
    - vu.utms.ui – Swing GUI classes, with one panel per module plus a dashboard.
    - vu.utms.util – helper for JDBC connection to Access.
- **Architecture Layers**
    - **Model Layer** – simple JavaBeans encapsulating fields and behaviors.
    - **DAO Layer** – isolates all SQL (Create, Read, Update, Delete).
    - **UI Layer** – Swing panels and dialogs for each module, plus login and dashboard.

---

3. OOP Principles in UTMS

Principle	Implementation Summary
(i) Abstraction via abstract classes/interfaces	• abstract class User defines common user attributes and the abstract requestTransport(Route r) without revealing subclass logic. • Interface VehicleOperations exposes service(), trackLocation(), scheduleRoute(...), hiding details of eventual implementations.
(ii) Encapsulation (private fields + getters/setters)	• All model fields (id, username, password, licenseNumber, etc.) are private. • Public getters/setters guard access and allow future validation.
(iii) Inheritance: base User and subclasses	• User contains shared fields/methods (login()). • Subclasses Student, Lecturer, TransportOfficer extend User, inheriting common behavior and adding role-specific fields (matricNumber, staffId).
(iv) Runtime polymorphism via method overriding	• Each subclass overrides requestTransport(Route r) to produce role-specific messages (e.g. "Lecturer...books priority seat"). • Code invokes user.requestTransport(r) on a User reference, and the correct subclass method runs at runtime.
(v) Interfaces for common vehicle behaviors	• VehicleOperations interface represents behaviors all vehicles share (service, tracking, scheduling). • The concrete Vehicle class implements these methods (printed stubs).
(vi) Method overriding of requestTransport()	• As noted above, requestTransport() is declared abstract in User and overridden in each subclass to reflect customized booking logic.
(vii) Method overloading of assignDriver()	• In Vehicle class, two overloaded methods: assignDriver(Driver d) assignDriver(Driver d, String shift)

	• Demonstrates compile-time polymorphism by invoking the appropriate overload based on arguments.

---

4. Key Design Choices

- **Lean Model Layer**
    - Combine all vehicle types into a single Vehicle class with a Type enum (BUS/VAN), rather than separate subclasses, to simplify the prototype while still illustrating interface implementation.
- **DAO Pattern**
    - Encapsulates all JDBC code in RouteDAO, DriverDAO, VehicleDAO, and UserDAO.
    - Each DAO offers full **CRUD** (Create, Read, Update, Delete) plus specialized operations like assignDriver(…) or authenticate(…).
- **Swing UI with Modular Panels**
    - **LoginUI** handles authentication.
    - **MainUI** hosts a JTabbedPane:
        - **HomeUI** shows a dashboard of record counts.
        - **RouteUI**, **DriverUI**, **VehicleUI**, **UserUI** each offer a JTable plus **Add/Edit/Delete/Refresh** buttons and dialogs for data entry.
    - Each panel is self-contained in its own class for clarity and maintainability.
- **Database Connection**
    - DatabaseConnection utility uses UCanAccess JDBC driver to connect to UTMS.accdb in a relative path, ensuring portability.

---

5. Demonstration Artifacts

- **Source Code**: All .java files are organized under vu.utms.* packages.
- **Screenshots**:
    - Console output showing overridden vs. overloaded method calls.

    - Each GUI panel: dashboard, routes table, drivers table, vehicles table, users table.
    - Add/Edit/Delete dialogs in action.
- **README**: Brief instructions on setting up UCanAccess, compiling in NetBeans, and running Main.java.

---

6. Conclusion

This UTMS prototype cleanly applies core OOP principles abstraction, encapsulation, inheritance, and polymorphism while demonstrating interface usage and method overloading. The layered DAO/UI architecture ensures separation of concerns, and the modular Swing interface provides a clear, maintainable structure suitable for group coursework.