### University Transport Management System (UTMS)

We were asked to create a Java program that shows how **Object-Oriented Programming (OOP)** works. We created a **University Transport Management System** that simulates how students, lecturers, and transport officers request vehicles like buses and vans.

The goal was to show how OOP concepts are used in a real-life example Project

We organized the code into different **packages** to keep things neat

* **Users package** contains abstract User class and subclasses: Student, Lecturer, TransportOfficer
* **Vehicle package** contains abstract Vehicle class and subclasses: Bus, Van
* **vehicle.interfaces package** contains interfaces: Serviceable, Schedulable, Trackable
* **Main.java** main class runs the program
* **TransportService.java class** handles driver assignment with method overloading

### OOP concepts we used

#### We used a****bstraction concept****

We created a class called **User** which only had the idea of what all users can do. But we didn’t fully define the method inside it . we left other classes like **Student** and **Lecturer** define their own behavior.

public abstract class User {

public abstract void requestTransport();

}

#### WE used e****ncapsulation****

We made the data inside each class private, meaning no one can access it directly. Instead, we used methods like **getName()** to access it safely.

private String name;public String getName() { return name; }

This keeps the data secure and organized.

#### We used ****Inheritance****

Classes like **Student** and **Lecturer** get properties from the main **User** class. And also, **Bus** and **Van** get their features from a **Vehicle** class. This helps reduce repeating code.

public class Student extends User { ... }

#### We used ****Polymorphism Overriding method****

Even though we used the same method name **requestTransport()** for all users, each one behaved differently.

Student said "Student Clark requested transport."

Lecturer said "Lecturer Dr. Bazigu requested academic transport."

Officer said "Transport Officer Mr. James is scheduling transport."

This is **runtime polymorphism, this** program decides which version to run based on the object.

#### We used ****Interfaces****

We created small contract-like files called interfaces for vehicles like **Trackable, Schedulable**, and **Serviceable**. Both **Bus** and **Van** had to follow these behaviors.

public class Bus implements Trackable, Schedulable, Serviceable { ... }

#### We used m****ethod overriding****

**requestTransport()** was defined differently in each **user** class. This is overriding.

Also, both **Bus** and **Van** had their own version of methods like track() and service().

#### ****We used method overloading****

We made two **assignDriver()** methods here

assignDriver("John", "Bus");

assignDriver("Linda", "10:00 PM", true);

Both have the same name but different parameters. This is called **overloading**.

### How we tested it

In **Main.java**, we created **user** objects and **vehicle** objects, then called their methods. We saw different outputs depending on the object. For example:

Student Clark requested transport.

Lecturer Dr. Bazigu requested academic transport.

...

This proved that OOP was working properly.