```java
// Abstraction:                abstract class
abstract class Vehicle {
    // Encapsulation: model              capacity
    private String model;
    private int capacity;

    // Constructor:
    public Vehicle(String model, int capacity) {
        setModel(model);          // Encapsulated setter call
        setCapacity(capacity);    // Encapsulated setter call
    }

    // Getter method: encapsulated
    public String getModel() {
        return model;
    }

    // Setter method: encapsulated
    public void setModel(String model) {
        if (!model.isEmpty()) {
            this.model = model;
        }
    }

    public int getCapacity() {
        return capacity;
    }

    public void setCapacity(int capacity) {
        if (capacity > 0) {
            this.capacity = capacity;
        }
    }

    // Abstract Method:
    public abstract void startEngine();
    public abstract void showDetails();
}

// Inheritance: Truck                              Vehicle
class Truck extends Vehicle {
    private double load; // Truck-specific

    // Constructor: Truck
    public Truck(String model, int capacity, double load) {
        super(model, capacity);  // Parent class constructor call
        this.load = load;
```

```
    }

    // Polymorphism: Vehicle class-       method override
    @Override
    public void startEngine() {
        System.out.println("Truck engine started with loud diesel sound!")
    }

    @Override
    public void showDetails() {
        System.out.println("    Truck Model: " getModel());
        System.out.println("    Capacity: " getCapacity() + " tons");
        System.out.println("    Current Load: " load + " tons");
    }

    // Truck-specific Method: Truck-
    public void loadGoods(double weight) {
        if (weight + load <= getCapacity()) {
            load += weight;
            System.out.println("    " weight + " tons loaded. Total Load:
        } else {
            System.out.println("    Overload! Cannot load " weight + " ton
        }
    }
}

// Interface:                                abstraction,              m
interface Maintainable {
    void performMaintenance(); // Method signature only
}

// Bus class:                                Vehicle, Maintainable interf
class Bus extends Vehicle implements Maintainable {
    private int passengers;

    public Bus(String model, int capacity, int passengers) {
        super(model, capacity);
        this.passengers = passengers;
    }

    @Override
    public void startEngine() {
        System.out.println("Bus engine started smoothly.");
    }

    @Override
    public void showDetails() {
        System.out.println("    Bus Model: " getModel());
```

```
        System.out.println("    Passenger Capacity: " + getCapacity());
        System.out.println("    Current Passengers: " + passengers);
    }

    // Interface        method
    @Override
    public void performMaintenance() {
        System.out.println("Bus maintenance scheduled monthly.");
    }
}

// Main                 :
public class Main {
    public static void main(String[] args) {
        // Polymorphism: Parent               Vehicle,
        Vehicle truck = new Truck("Volvo FMX", 20, 5);
        Vehicle bus = new Bus("Hyundai Super", 40, 25);

        truck.startEngine();    // Polymorphic behavior
        truck.showDetails();

        System.out.println();

        bus.startEngine();      // Polymorphic behavior
        bus.showDetails();

        System.out.println();

        // Type casting: Truck-specific method call
        if (truck instanceof Truck) {
            Truck t = (Truck) truck;
            t.loadGoods(10);   //
            t.loadGoods(7);    //
        }

        System.out.println();

        // Interface                                  maintenance
        if (bus instanceof Maintainable) {
            Maintainable m = (Maintainable) bus;
            m.performMaintenance();
        }
    }
}
```