

OOP_RyanEvans_2420520

by Ryan Evans

Submission date: 25-Apr-2025 11:40AM (UTC+0100)

Submission ID: 256592101

File name: 58749_Ryan_Evans_OOP_RyanEvans_2420520_662765_1501402654.docx (1.55M)

Word count: 4668

Character count: 36186

Object-Oriented Car Park Simulation

User Guide

1. Clone/download/unzip the project.
2. Ensure JDK 17+ is installed (Corretto 23 recommended).
3. Generate Maven sources
4. Run the main class: `CarParkSimulation.java`
5. Output is displayed in the console
6. Logs are generated in `src/logs/`
7. Configurable parameters are in `config.json`
8. To enable persistent vehicle storage, comment out line 29 (`clearVehicleFile()`) in `VehicleData`. (It is currently enabled due to running the simulation over the same 7-day period, which results in negative payments).

Table of Contents	
Best Practices in OOP Implementation	3
Aggregation	3
Inheritance	4
Abstract Classes	4
polymorphism	6
Interfaces	6
Design Patterns (Observer Pattern)	6
Class Descriptions and Relationships	7
CarParkManager	7
VehicleEntryHandler	7
VehicleExitHandler	7
CarPark	8
VehicleData	8
VehicleInfo	8
Barrier (abstract)	8
Sensor (interface)	9
IDReader (abstract)	9
FullSign	9
Logger	9
Config & ConfigLoader	10
PaymentCalculator	10
ProbabilityScheduler	10
Static vs Dynamic Memory Allocation	11
Static Memory Allocation	11
Dynamic Memory Allocation	11
Justification of Data Structure Choices	11
HashMap<String, String> and HashMap<String, VehicleInfo>	11
HashSet<String>	11
List<Observer> (via ArrayList)	12
ArrayList<VehicleInfo>	12
JSON Files (via List<VehicleInfo> serialised)	12
Testing	13
Testing Frameworks and Tools	13
Testing Objectives	13
Key Unit Test Examples	13
PaymentCalculatorTest	13
VehicleEntryHandler	14
VehicleExitHandler	14
Test Harness and Simulation Control	14
Video Demonstration	16
Appendix	17
Generated Files	17
Source Code	17
References	34

Best Practices in OOP Implementation

This section examines the implementation of key (OOP) object orientated programming principles within the car park simulation. Primarily on Aggregation inheritance, polymorphism and design patterns.

Aggregation

Within the simulation, the CarParkManager class aggregates multiple components, including the VehicleEntryHandler, VehicleExitHandler, CarPark, and Logger classes, and includes various readers, sensors and barriers. The individual components encapsulate their own specific responsibilities, such as managing vehicle entry and exit, logging events, and processing number plate and barcode information, while remaining loosely coupled with the manager class. This promotes modularity [1] With the ability to independently modify or add new components to the system without the need to modify the manager class.

1

```
public void handleEntry(String timestamp, Logger logger, String regNumber,
                        String type, String staffBarcode) throws IOException {
    entranceSensor.detectCar();
    logger.log(timestamp + " - Vehicle detected by Entrance Sensor");

    if (carPark.getFreeSpaces() == 0) {
        displayFullCarArt();
        logger.log(timestamp + " - CarPark is Full");
        return;
    }

    if (numberPlateReader.validateNumberplate(regNumber)) {
        logger.log(timestamp + " - Valid number plate " + regNumber);
        entranceBarrier.openBarrier();
        entranceBarrier.updateBarrier();
        logger.log(timestamp + " - Entrance barrier opened");

        vehicleData.addVehicle(regNumber, timestamp, type, staffBarcode);
        carPark.setOccupiedSpaces(carPark.getOccupiedSpaces() + 1);

        logger.log(timestamp + " - " + type + " entered with Registration: "
                    + regNumber + " | Remaining spaces: " + carPark.getFreeSpaces());

        entranceBarrier.closeBarrier();
        entranceBarrier.updateBarrier();
        logger.log(timestamp + " - Entrance Barrier Closed");
    } else {
        logger.log(timestamp + " - Invalid number plate " + regNumber);
    }
}
```

2

```

public void handleExit(String timestamp, Logger logger, String regNumber,
                      String type, String staffBarcode, double cost) throws IOException {
    exitSensor.detectCar();
    logger.log(timestamp + " - Vehicle detected by Exit Sensor");

    logger.log(timestamp + " - " + regNumber + " Payment Calculated at £" + cost);
    logger.log(timestamp + " - " + regNumber + " Payment made");

    exitBarrier.openBarrier();
    exitBarrier.updateBarrier();
    logger.log(timestamp + " - Exit Barrier Opened");

    carPark.setOccupiedSpaces(carPark.getOccupiedSpaces() - 1);

    if (staffBarcode != null) {
        logger.log(timestamp + " - " + type + ": " + staffBarcode + " exited with Registration: "
                  + regNumber + " | Remaining spaces: " + carPark.getFreeSpaces());
    } else {
        logger.log(timestamp + " - " + type + " exited with Registration: "
                  + regNumber + " | Remaining spaces: " + carPark.getFreeSpaces());
    }

    exitBarrier.closeBarrier();
    exitBarrier.updateBarrier();
    logger.log(timestamp + " - Exit Barrier Closed");

    vehicleData.markVehicleExited(regNumber, timestamp);
}
}

```

```

public boolean validateNumberplate(String regNumber) {
    Pattern pattern = Pattern.compile("^[A-Z]{2}\\d{2} [A-Z]{3}$");
    Matcher matcher = pattern.matcher(regNumber);

    if (matcher.matches()) {
        System.out.println("Valid Number Plate");
        return true;
    }
    else{
        System.out.println("Invalid Number Plate");
        return false;
    }
}

```

Inheritance

Within the carpark simulation, inheritance has been employed to promote code reusability and extensibility. [2]

Abstract Classes

The IDReader is an abstract class that defines the behaviour of recognition mechanisms. It is extended by the concrete classes BarcodeReader and NumberPlateReader, each implement their own device specific logic.

```

package CarPark.Devices.IDReaders;

import java.io.FileNotFoundException;

public abstract class IdReader {
    public abstract String read() throws FileNotFoundException;
}

```

```

@Override
public String read() throws FileNotFoundException {
    JSONParser parser = new JSONParser();
    InputStream inputStream = getClass().getClassLoader().getResourceAsStream("staff.json");
    if (inputStream == null) {
        throw new FileNotFoundException("staff.json not found in resources");
    }
    try (InputStreamReader reader = new InputStreamReader(inputStream)) {
        JSONArray staffNumber = (JSONArray) parser.parse(reader);
        for (Object o : staffNumber) {
            JSONObject staffObj = (JSONObject) o;
            if (regNumber.equals(staffObj.get("regNumber"))) {
                if (staffObj.get("staffBarcode").equals(null)) {
                    throw new NullPointerException("barcode not found in staff.json");
                } else {
                    System.out.println(regNumber + " " + staffObj.get("staffBarcode").toString());
                    return staffObj.get("staffBarcode").toString();
                }
            }
        }
    }

    } catch (IOException | ParseException e){
        throw new RuntimeException(e);
    }
    return null;
}

```

```

@Override
public String read() throws FileNotFoundException {
    JSONParser parser = new JSONParser();
    InputStream inputStream = getClass().getClassLoader().getResourceAsStream("uk_numberplates.json");
    if (inputStream == null) {
        throw new FileNotFoundException("uk_numberplates.json not found in resources");
    }

    try (InputStreamReader reader = new InputStreamReader(inputStream)) {
        JSONArray ukNumberplates = (JSONArray) parser.parse(reader);
        int index = (int)(Math.random() * ukNumberplates.size());
        JSONObject plateObj = (JSONObject) ukNumberplates.get(index);
        regNumber = (String) plateObj.get("regNumber");
        System.out.println("Number Plate: " + regNumber);
    } catch (IOException | ParseException e) {
        throw new RuntimeException(e);
    }
    return regNumber;
}

```

Similarly, the Barrier abstract class encapsulates common attributes and methods.[3] required by all physical barriers such as location and the open state. Whereas the associated subclasses EntranceBarrier and ExitBarrier, while currently not implemented, can have their own entry and exit logic if required.

4 polymorphism

This simulation design allows for polymorphic substitution, allowing components such as CarParkManager to interact with the barriers and ID readers through the abstract types without reliance on the concrete implementations. [4]. This allows for future expansion with the ability to add new reader types or different barriers, such as a staff only barrier, or QR code readers.

Interfaces

Interfaces are used within the simulation to enforce behavioural contracts, decoupling logic from implementation[5]. The sensor interface defines a single method to detect vehicles and is implemented by both entranceSensor and exitSensor. This allows the CarParkManager to interact with any sensor without knowing the specific details of how vehicles are detected. This ensures expansion opportunities for the system, allowing for new sensors to be implemented.

Design Patterns (Observer Pattern)

The simulation uses the observer design pattern to decouple the system's internal state from the user interface components. The CarPark class acts as the subject that prompts an update from the observer, which is the FullSign class. The car park occupancy triggers the observer and subsequently updates the display sign. The use of the observer in this context follows the single responsibility principle[6]. Separating the logic required to track occupancy also does not manage the display logic. It also allows for expandability[7], with the option to implement additional observers for future functionality.

```
@Override
public void update(boolean full, int freeSpaces) {
    this.full = full;
    this.freeSpaces = freeSpaces;
    print();
}
```

```
@Override
public void notifyObservers() {
    boolean full = occupiedSpaces >= capacity;
    int freeSpaces = capacity - occupiedSpaces;
    for (Observer o : observers) {
        o.update(full, freeSpaces);
    }
}
```

Class Descriptions and Relationships

CarParkManager

Responsibility: Acts as the central coordinator of the simulation. It manages the overall workflow, including vehicle entry and exit, time progression, logging, and coordination between subsystems.

Relationships:

- Aggregates VehicleEntryHandler, VehicleExitHandler, CarPark, Logger, Config, and probabilityScheduler.
- Receives data from readers BarcodeReader, NumberPlateReader and updates the car park state accordingly.

VehicleEntryHandler

Responsibility: Encapsulates the logic related to processing vehicles entering the car park, including sensor detection, barrier control, registration validation, and vehicle data registration.

Relationships:

- Interacts with Sensor, Barrier, VehicleData, and CarPark.
- Called by CarParkManager during simulated vehicle entries

VehicleExitHandler

Responsibility: Handles the exit process, including vehicle identification, payment calculation, barrier control, and updating vehicle status and car park occupancy.

Relationships:

- Uses Sensor, Barrier, VehicleData, and CarPark.
- Called by CarParkManager during simulation exits.

CarPark

Responsibility: Maintains the state of the parking facility, including capacity and occupancy. Implements the Subject interface to notify observers of state changes.

Relationships:

- Notifies FullSign (an Observer) of changes in occupancy.
- Accessed by handlers and the manager to determine fullness and update status.

VehicleData

Responsibility: Stores and manages both in-memory and persistent vehicle data, including registration numbers, timestamps, vehicle type, and parking status.

Relationships:

- Called by handlers to add, remove, or query vehicle state.
- Serialises data to and from a JSON file.

VehicleInfo

Responsibility: Represents a complete record of a single vehicle's state, including entry and exit times, type (staff/customer), and barcode (if applicable).

Relationships:

- Stored in VehicleData for persistent tracking.

Barrier (abstract)

Responsibility: Defines the structure and shared behaviour of entry and exit barriers.

Subclasses: EntranceBarrier, ExitBarrier

Relationships:

- Used by entry/exit handlers to control vehicle flow.
- Each subclass manages a physical access point.

Sensor (interface)

Responsibility: Provides an abstraction for simulated hardware sensors that detect cars.

Implementations: EntranceSensor, ExitSensor

Relationships:

- Used polymorphically in handlers to trigger simulation behaviour upon vehicle detection.

IDReader (abstract)

Responsibility: Represents an abstract interface for any ID reading mechanism.

Subclasses: BarcodeReader, NumberPlateReader

Relationships:

- Used by the manager to retrieve registration numbers and determine vehicle types.
- Used by the manager to check if a valid staff barcode is presented.

FullSign

Responsibility: Implements the Observer interface and reflects the current state of the car park (e.g., full status and remaining spaces).

Relationships:

- Observes the CarPark for changes.
- Outputs updates visually to the console.

Logger

Responsibility: Handles persistent logging of simulation events to dated text files for later review or debugging.

Relationships:

- Used by CarParkManager and handlers to log timestamps, vehicle activity, and barrier states.

Config & ConfigLoader

Responsibility: Parses external configuration data (JSON) and exposes typed parameters such as pricing, capacity, and time-based probabilities.

Relationships:

- Injected into CarParkManager, PaymentCalculator, and ProbabilityScheduler to guide runtime behaviour.

PaymentCalculator

Responsibility: Calculates parking charges based on time duration and type (with discounts for staff).

Relationships:

- Used in VehicleExitHandler and CarParkManager during vehicle departure.

ProbabilityScheduler

Responsibility: Determines entry and exit probabilities at specific times of day, allowing the simulation to model real-world traffic patterns.

Relationships:

- Reads from Config, used by CarParkManager to decide whether to simulate entry or exit events each interval.

Static vs Dynamic Memory Allocation

Static Memory Allocation

Static memory is allocated during compilation, which reserves the memory required before the program starts. This is used for primitive data types, static variables at class level and constraints[8].

These variables are typically stored in the stack or method area[8]. Static memory allocation is fast and predictable but lacks flexibility as the size must be known and not require modification during execution.

Dynamic Memory Allocation

Dynamically allocated memory takes place during runtime and is used whenever a new object is created and for dynamic data structures such as ArrayLists, HashMaps, and HashSets[9].

Dynamically allocated memory is stored on the heap and is managed via a garbage collector[9]. Java provides a garbage collector which reclaims memory when an object is no longer required. This provides flexibility and scalability while the program is running.

Justification of Data Structure Choices

The simulation has multiple data structures implemented to perform different tasks. It uses HashMaps, HashSets, ArrayLists, Lists, and JSON files. I primarily use dynamic memory as it allows for an undetermined number of vehicles to be simulated[9]. It also allows for fast $O(1)$ lookups[10].

HashMap<String, String> and HashMap<String, VehicleInfo>

Used in: VehicleData

Purpose:

- Stores currently parked vehicles with registration numbers as keys and either entry timestamps or full VehicleInfo objects as values.

Why it's appropriate:

- Provides $O(1)$ time complexity for lookups, insertions, and deletions.
- Perfect for checking if a vehicle is already parked or retrieving entry time instantly.

HashSet<String>

Possible Use: Tracking known registration numbers or preventing duplicate entries.

Why it's appropriate:

- Guarantees the **uniqueness** of entries.
- Offers fast membership checks (like contains()) in $O(1)$.

List<Observer> (via ArrayList)

Used in: CarPark (implements Subject)

Purpose:

- Stores registered Observer instances (like FullSign) to notify them of state changes.

Why it's appropriate:

- A simple and efficient way to support multiple subscribers.
- Ensures predictable iteration order.

ArrayList<VehicleInfo>

Used in: VehicleData

Purpose:

- Maintains a complete list of all vehicles that have ever entered the car park, regardless of status.

Why it's appropriate:

- Supports **dynamic resizing** and ordered traversal.
- Enables exporting, logging, or iterating through historical vehicle records.

JSON Files (via List<VehicleInfo> serialised)

Used for: Persistent storage in vehicles.json

Purpose:

- Persists full vehicle history to disk.
- Backed by an in-memory ArrayList but stored as JSON.

Why it's appropriate:

- Lightweight and human-readable.
- Integrates well with the load/save logic.

Testing

Testing Frameworks and Tools

The simulation uses industry standard testing frameworks, ensuring correctness, maintainability and regression safety[11].

JUnit 5 – Provides the foundation for writing and executing unit tests.

Mockito (v5.14.2) – Enables mocking of dependencies, such as sensors and barriers, to isolate the unit under test.

Hamcrest – Used for readable assertions (e.g., `assertThat(...)`).

Testing Objectives

- Business logic correctness (vehicle entry, exit, pricing)
- Object interactions via interfaces and abstract classes
- Configuration parsing and system behaviour under variable conditions
- Realistic time-based scenarios through simulation intervals

Key Unit Test Examples

PaymentCalculatorTest

Verifies that staff discounts are applied correctly over a 2-hour interval

```
@Test
void appliesStaffDiscountCorrectly() {
    PaymentCalculator calc = new PaymentCalculator(1.25, 0.1);

    String entryTime = "2025-06-01T10:00:00";
    String exitTime = "2025-06-01T12:00:00";

    double result = calc.calculate("Staff", entryTime, exitTime);
    assertEquals(0.25, result, 0.01); // 2h * 1.25 * 0.1 = 0.25
}
```

VehicleEntryHandler

Confirms that entry logic utilises the correct components and records the vehicle

```
@Test
void allowsEntryWhenPlateIsValid() throws IOException {
    String regNumber = "ABC123";
    String timestamp = "2025-06-01T10:00:00";
    when(numberPlateReader.validateNumberplate(regNumber)).thenReturn(true);

    entryHandler.handleEntry(timestamp, logger, regNumber, "Customer", null);

    verify(sensor).detectCar();
    verifyNoMoreInteractions(sensor);
    verify(barrier).openBarrier();
    verify(barrier).closeBarrier();
    verify(vehicleData).addVehicle(eq(regNumber), any(), eq("Customer"), isNull());

    assert carPark.getOccupiedSpaces() == 11;
}
```

VehicleExitHandler

Confirms that the exit logic executes and logs are created.

```
@Test
void testValidExitLogsAndUpdatesOccupancy() throws IOException {
    String regNumber = "XYZ123";
    String timestamp = "2025-06-01T18:00:00";
    String type = "Customer";
    String staffBarcode = null;
    double cost = 4.50;

    exitHandler.handleExit(timestamp, logger, regNumber, type, staffBarcode, cost);

    verify(sensor).detectCar();
    verify(barrier).openBarrier();
    verify(barrier).closeBarrier();

    verify(logger).log(contains("detected by Exit Sensor"));
    verify(logger).log(contains("Payment Calculated"));
    verify(logger).log(contains("Payment made"));
    verify(logger).log(contains("Exit Barrier Closed"));

    verify(vehicleData).markVehicleExited(regNumber, timestamp);

    assert carPark.getOccupiedSpaces() == 9;
}
```

Test Harness and Simulation Control

The runWeeklySimulation method in the CarparkManager itself acts as a test harness.

The simulation confirms the integration of subsystems and supports exploratory testing of various real-world patterns.

- It simulates a 7-day period with 15-minute intervals.
- It randomly triggers vehicle entries and exits based on the probabilityScheduler
- It generated daily log files of activity

```
2025-06-02T12:00:00 - Valid number plate MN19 MGJ
2025-06-02T12:00:00 - Entrance barrier opened
2025-06-02T12:00:00 - Customer entered with Registration: MN19 MGJ | Remaining spaces: 84
2025-06-02T12:00:00 - Entrance Barrier Closed
2025-06-02T12:15:00 - Vehicle detected by Exit Sensor
2025-06-02T12:15:00 - KW62 XTH Payment Calculated at £2.81
2025-06-02T12:15:00 - KW62 XTH Payment made
2025-06-02T12:15:00 - Exit Barrier Opened
2025-06-02T12:15:00 - Customer: null exited with Registration: KW62 XTH | Remaining spaces: 85
```

```
Car detected: Entrance

      -----
      /|_|_|_|_\'.--
      (  _  _  _ \
      =`-(_)--( )-`
Valid Number Plate
Barrier Entrance is now open.

| NPC:           Swansea |
|                 |
| Car park full:    X   |
|                 |
| Free spaces:      99   |
|                 |
Barrier Entrance is now closed.
```

Video Demonstration

<https://youtu.be/S1xAyzROhnY>





Appendix

Generated Files

Mockaroo was used to generate a JSON file to contain 1000 number plates with 100 invalid number plates.

Source Code

```
package CarPark;

import CarPark.Manager.CarParkManager;

public class CarParkSimulation {
    public static void main(String[] args) {
        CarParkManager manager = new CarParkManager();
        manager.runWeeklySimulation();
    }
}
```

```
package CarPark.Config;

import java.util.Map;

public class Config {
    private double pricePerHour;
    private double discountFactorStaff;
    private int capacity;
    private Map<String, TimeWindowProbability> entryExitProbabilities;

    public double getPricePerHour() {
        return pricePerHour;
    }

    public void setPricePerHour(double pricePerHour) {
        this.pricePerHour = pricePerHour;
    }

    public double getDiscountFactorStaff() {
        return discountFactorStaff;
    }

    public void setDiscountFactorStaff(double discountFactorStaff) {
        this.discountFactorStaff = discountFactorStaff;
    }

    public int getCapacity() {
        return capacity;
    }

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

    public Map<String, TimeWindowProbability> getEntryExitProbabilities() {
        return entryExitProbabilities;
    }

    public static class TimeWindowProbability {
```

```

        private int startHour;
        private int endHour;
        private double entry;
        private double exit;

        public int getStartHour() {
            return startHour;
        }

        public int getEndHour() {
            return endHour;
        }

        public double getEntry() {
            return entry;
        }

        public double getExit() {
            return exit;
        }
    }
}

```

```

package CarPark.Config;

import com.fasterxml.jackson.databind.ObjectMapper;
import java.io.InputStream;

public class ConfigLoader {

    public static Config load(String fileName) {
        try {
            ObjectMapper mapper = new ObjectMapper();
            InputStream inputStream =
                ConfigLoader.class.getClassLoader().getResourceAsStream("config.json");
            if (inputStream == null) {
                throw new RuntimeException("Unable to load config file: " +
                    fileName);
            }
            return mapper.readValue(inputStream, Config.class);
        } catch (Exception e) {
            throw new RuntimeException("Unable to load config file: " + fileName,
                e);
        }
    }
}

```

```

package CarPark.Devices.Barriers;

public abstract class Barrier {
    protected String location;
    protected boolean open;

    public Barrier(String location) {
        this.location = location;
        this.open = false;
    }

    public abstract void openBarrier();

    public abstract void closeBarrier();

    public void updateBarrier() {

```

```

        System.out.println("Barrier " + location + " is now " + (open ? "open" :
"closed") + ".");
    }

    public String getLocation() {
        return location;
    }
}

```

```

package CarPark.Devices.Barriers;

public class EntranceBarrier extends Barrier {

    public EntranceBarrier(String Location) {
        super(Location);
    }

    @Override    public void openBarrier() {
        open = true;
    }

    @Override    public void closeBarrier() {
        open = false;
    }
}

```

```

package CarPark.Devices.Barriers;

public class ExitBarrier extends Barrier {

    public ExitBarrier(String Location) {
        super(Location);
    }

    @Override    public void openBarrier() {
        open = true;
    }

    @Override    public void closeBarrier() {
        open = false;
    }
}

```

```

package CarPark.Devices.IDReaders;

import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

public class BarcodeReader extends IdReader {
    private String regNumber;

    public String getRegNumber() {
        return regNumber;
    }
}

```

```

    }

    public void setRegNumber(String regNumber) {
        this.regNumber = regNumber;
    }

    @Override public String read() throws FileNotFoundException {
        JSONParser parser = new JSONParser();
        InputStream inputStream =
getClass().getClassLoader().getResourceAsStream("staff.json");
        if (inputStream == null) {
            throw new FileNotFoundException("staff.json not found in resources");
        }
        try (InputStreamReader reader = new InputStreamReader(inputStream)) {
            JSONArray staffNumber = (JSONArray) parser.parse(reader);
            for (Object o : staffNumber) {
                JSONObject staffObj = (JSONObject) o;
                if (regNumber.equals(staffObj.get("regNumber"))) {
                    if (staffObj.get("staffBarcode").equals(null)) {
                        throw new NullPointerException("barcode not found in
staff.json");
                    } else {
                        System.out.println(regNumber + " " +
staffObj.get("staffBarcode").toString());
                        return staffObj.get("staffBarcode").toString();
                    }
                }
            }
        } catch (IOException | ParseException e) {
            throw new RuntimeException(e);
        }
        return null;
    }
}

```

```

package CarPark.Devices.IDReaders;

import java.io.FileNotFoundException;

public abstract class IdReader {
    public abstract String read() throws FileNotFoundException;
}

```

```

package CarPark.Devices.IDReaders;

import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import java.io.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class NumberPlateReader extends IdReader {
    private String regNumber;

    public void setRegNumber(String regNumber) {
        this.regNumber = regNumber;
    }
}

```

```

    }

    @Override    public String read() throws FileNotFoundException {
        JSONParser parser = new JSONParser();
        InputStream inputStream =
getClass().getClassLoader().getResourceAsStream("uk_numberplates.json");
        if (inputStream == null) {
            throw new FileNotFoundException("uk_numberplates.json not found in
resources");
        }

        try (InputStreamReader reader = new InputStreamReader(inputStream)) {

            JSONArray ukNumberplates = (JSONArray) parser.parse(reader);
            int index = (int) (Math.random() * ukNumberplates.size());
            JSONObject plateObj = (JSONObject) ukNumberplates.get(index);
            regNumber = (String) plateObj.get("regNumber");
            System.out.println("Number Plate: " + regNumber);
        } catch (IOException | ParseException e) {
            throw new RuntimeException(e);
        }
        return regNumber;
    }

    public boolean validateNumberplate(String regNumber) {
        Pattern pattern = Pattern.compile("[A-Z]{2}\\d{2} [A-Z]{3}$");
        Matcher matcher = pattern.matcher(regNumber);

        if (matcher.matches()) {
            System.out.println("Valid Number Plate");
            return true;
        }
        else{
            System.out.println("Invalid Number Plate");
            return false;
        }
    }
}

```

```

package CarPark.Devices.Sensors;

public interface Sensor {
    void detectCar();
}

```

```

package CarPark.Devices.Sensors;

public class EntranceSensor implements Sensor {

    @Override    public void detectCar() {
        System.out.println("Car detected: Entrance");
        System.out.println("_____\\n" +
            " /|_|_|\\|\\`._\\n" +
            " |( _|_|_|\\|\\|\\n" +
            " ='-( _)--( _)-' ");
    }
}

```

```

package CarPark.Devices.Sensors;

public class ExitSensor implements Sensor {

```

1

3

```

        NumberPlateReader numberPlateReader, VehicleData
vehicleData, CarPark carPark) {
    this.entranceSensor = entranceSensor;
    this.entranceBarrier = entranceBarrier;
    this.numberPlateReader = numberPlateReader;
    this.vehicleData = vehicleData;
    this.carPark = carPark;
}

public void handleEntry(String timestamp, Logger logger, String regNumber,
                        String type, String staffBarcode) throws IOException {
    entranceSensor.detectCar();
    logger.log(timestamp + " - Vehicle detected by Entrance Sensor");

    if (carPark.getFreeSpaces() == 0) {
        displayFullCarArt();
        logger.log(timestamp + " - CarPark is Full");
        return;
    }

    if (numberPlateReader.validateNumberplate(regNumber)) {
        logger.log(timestamp + " - Valid number plate " + regNumber);
        entranceBarrier.openBarrier();
        entranceBarrier.updateBarrier();
        logger.log(timestamp + " - Entrance barrier opened");

        vehicleData.addVehicle(regNumber, timestamp, type, staffBarcode);
        carPark.setOccupiedSpaces(carPark.getOccupiedSpaces() + 1);

        logger.log(timestamp + " - " + type + " entered with Registration: "
+ regNumber + " | Remaining spaces: " + carPark.getFreeSpaces());

        entranceBarrier.closeBarrier();
        entranceBarrier.updateBarrier();
        logger.log(timestamp + " - Entrance Barrier Closed");
    } else {
        logger.log(timestamp + " - Invalid number plate " + regNumber);
    }
}

private void displayFullCarArt() {
    System.out.println("Carpark is Full");
    System.out.println("      _____\n" +
        "    _.'/_|_|_|\ \ \ \n" +
        "  /_/_)\n" +
        "'-(_)--( )-=");
}
}

```

```

package CarPark.Handlers;

import CarPark.Devices.Barriers.Barrier;
import CarPark.Devices.Sensors.Sensor;
import CarPark.IO.Logger;
import CarPark.Models.CarPark;
import CarPark.Models.VehicleData;

import java.io.IOException;

public class VehicleExitHandler {

```

```

private final Sensor exitSensor;
private final Barrier exitBarrier;
private final VehicleData vehicleData;
private final CarPark carPark;

public VehicleExitHandler(Sensor exitSensor, Barrier exitBarrier,
                          VehicleData vehicleData, CarPark carPark){
    this.exitSensor = exitSensor;
    this.exitBarrier = exitBarrier;
    this.vehicleData = vehicleData;
    this.carPark = carPark;
}

public void handleExit(String timestamp, Logger logger, String regNumber,
                      String type, String staffBarcode, double cost)
throws IOException {
    exitSensor.detectCar();
    logger.log(timestamp + " - Vehicle detected by Exit Sensor");

    logger.log(timestamp + " - " + regNumber + " Payment Calculated at £" +
cost);
    logger.log(timestamp + " - " + regNumber + " Payment made");

    exitBarrier.openBarrier();
    exitBarrier.updateBarrier();
    logger.log(timestamp + " - Exit Barrier Opened");

    carPark.setOccupiedSpaces(carPark.getOccupiedSpaces() - 1);

    if (staffBarcode != null) {
        logger.log(timestamp + " - " + type + ": " + staffBarcode + "
exited with Registration: " + regNumber + " | Remaining spaces: " +
carPark.getFreeSpaces());
    } else {
        logger.log(timestamp + " - " + type + " exited with Registration: "
+ regNumber + " | Remaining spaces: " + carPark.getFreeSpaces());
    }

    exitBarrier.closeBarrier();
    exitBarrier.updateBarrier();
    logger.log(timestamp + " - Exit Barrier Closed");

    vehicleData.markVehicleExited(regNumber, timestamp);
}
}

```

```

package CarPark.IO;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class Logger {
    private final BufferedWriter writer;

    public Logger(String logFilePath) throws IOException {
        this.writer = new BufferedWriter(new FileWriter(logFilePath));
    }

    public void log(String message) throws IOException {
        writer.write(message);
        writer.newLine();
    }

    public void close() throws IOException {

```

```

        writer.flush();
        writer.close();
    }
}

```

```

package CarPark.Logic;

import java.time.LocalDateTime;
import java.time.temporal.ChronoUnit;

public class PaymentCalculator {
    private final double baseRate;
    private final double staffDiscount;

    public PaymentCalculator(double baseRate, double staffDiscount) {
        this.baseRate = baseRate;
        this.staffDiscount = staffDiscount;
    }

    public double calculate(String type, String entryTimeStr, String exitTimeStr) {
        double rate = "Staff".equals(type) ? baseRate * staffDiscount : baseRate;
        LocalDateTime entry = LocalDateTime.parse(entryTimeStr);
        LocalDateTime exit = LocalDateTime.parse(exitTimeStr);
        long minutes = ChronoUnit.MINUTES.between(entry, exit);
        double duration = minutes / 60.0;
        return Math.round(rate * duration * 100.0) / 100.0;
    }
}

```

```

package CarPark.Logic;

import CarPark.Config.Config;

import java.util.Map;

public class ProbabilityScheduler {

    private final Config config;

    public ProbabilityScheduler(Config config) {
        this.config = config;
    }

    public double getProbability(int hour, boolean isEntry) {
        for (Map.Entry<String, Config.TimeWindowProbability> entry :
config.getEntryExitProbabilities().entrySet()) {
            Config.TimeWindowProbability window = entry.getValue();
            boolean inRange = (window.getStartHour() <= window.getEndHour())
                ? hour >= window.getStartHour() && hour < window.getEndHour()
                : hour >= window.getStartHour() || hour < window.getEndHour();
            // handles overnight
            if (inRange) {
                return isEntry ? window.getEntry() : window.getExit();
            }
        }
        Config.TimeWindowProbability def =
config.getEntryExitProbabilities().get("default");
        return isEntry ? def.getEntry() : def.getExit();
    }
}

```

```

package CarPark.Manager;

import CarPark.Config.Config;
import CarPark.Config.ConfigLoader;
import CarPark.Devices.Barriers.ExitBarrier;
import CarPark.Devices.IDReaders.BarcodeReader;
import CarPark.Devices.Sensors.ExitSensor;
import CarPark.Devices.Sign.FullSign;
import CarPark.Devices.IDReaders.NumberPlateReader;
import CarPark.Handlers.VehicleEntryHandler;
import CarPark.Handlers.VehicleExitHandler;
import CarPark.IO.Logger;
import CarPark.Models.CarPark;
import CarPark.Devices.Barriers.Barrier;
import CarPark.Devices.Barriers.EntranceBarrier;
import CarPark.Devices.Sensors.EntranceSensor;
import CarPark.Models.VehicleData;
import CarPark.Logic.ProbabilityScheduler;
import CarPark.Logic.PaymentCalculator;

import java.io.*;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class CarParkManager {
    private final Config config = ConfigLoader.load("config.json");
    private final VehicleData vehicleData = new VehicleData();
    private final CarPark carPark = new CarPark();
    private final FullSign fullSign;
    private final NumberPlateReader numberPlateReader = new NumberPlateReader();
    private final BarcodeReader barcodeReader = new BarcodeReader();
    private final EntranceSensor entranceSensor = new EntranceSensor();
    private final ExitSensor exitSensor = new ExitSensor();
    private final Barrier entranceBarrier = new EntranceBarrier("Entrance");
    private final Barrier exitBarrier = new ExitBarrier("Exit");
    private final String location = "Swansea";
    private final ProbabilityScheduler scheduler = new
ProbabilityScheduler(config);
    private final PaymentCalculator paymentCalculator;
    private final VehicleEntryHandler entryHandler;
    private final VehicleExitHandler exitHandler;
    private int capacity;
    private double pricePerHour;
    private double staffDiscountFactor;

    public CarParkManager() {
        this.capacity = config.getCapacity();
        this.pricePerHour = config.getPricePerHour();
        this.staffDiscountFactor = config.getDiscountFactorStaff();

        fullSign = new FullSign(false, capacity, location);
        paymentCalculator = new PaymentCalculator(pricePerHour,
staffDiscountFactor);
        entryHandler = new VehicleEntryHandler(entranceSensor, entranceBarrier,
numberPlateReader, vehicleData, carPark);
        exitHandler = new VehicleExitHandler(exitSensor, exitBarrier, vehicleData,
carPark);
        carPark.setCapacity(capacity);
        carPark.setLocation(location);
        carPark.addObserver(fullSign);
        carPark.setPricePerHour(pricePerHour);
    }

    public void runWeeklySimulation() {

```

```

        LocalDateTime currentTime = LocalDateTime.of(2025, 6, 1, 0, 0);
        DateTimeFormatter timestampFormatter =
DateTimeFormatter.ISO_LOCAL_DATE_TIME;
        DateTimeFormatter logFileFormatter = DateTimeFormatter.ofPattern("yyyy-MM-
dd");

        new File("src/logs").mkdir();

        for (int day = 0; day < 7; day++) {
            String dateString = currentTime.format(logFileFormatter);
            String logFilePath = "src/logs/log_" + dateString + ".txt";

            Logger logger = null;
            try {
                logger = new Logger(logFilePath);
                logger.log(location + "\t" + "Car Park Logs for: " + "\t" +
dateString);

                for (int interval = 0; interval < 96; interval++) {
                    currentTime = currentTime.plusMinutes(15);
                    String timestamp = currentTime.format(timestampFormatter);

                    fullSign.update(isFull(), carPark.getFreeSpaces());

                    int hour = currentTime.getHour();
                    double entryProbability = scheduler.getProbability(hour, true);
                    double exitProbability = scheduler.getProbability(hour, false);

                    double action = Math.random();
                    if (action < entryProbability) {
                        processVehicleEntry(timestamp, logger);
                    } else if (action < entryProbability + exitProbability &&
!isEmpty()) {
                        processVehicleExit(timestamp, logger);
                    }

                    logger.log("End of log for " + dateString);
                } catch (IOException e) {
                    e.printStackTrace();
                } finally {
                    if (logger != null) {
                        try {
                            logger.close();
                        } catch (IOException
e) {
                            e.printStackTrace();
                        }
                    }
                }
            }
        }

        private void processVehicleEntry(String timestamp, Logger logger) throws
IOException {
            String regNumber = generateUniquePlate();
            String[] staffInfo = checkStaff(regNumber).split("\\|");
            String type = staffInfo[0];
            String staffBarcode = staffInfo[1];
            entryHandler.handleEntry(timestamp, logger, regNumber, type, staffBarcode);
        }

        private void processVehicleExit(String timestamp, Logger logger) throws
IOException {
            String regNumber = vehicleData.peekRandomVehicle();
            if (regNumber == null) {

```

```

        System.out.println("No Record of Licence Plate Detected - please
contact member of staff");
        return;
    }
    String[] staffInfo = checkStaff(regNumber).split("\\|");
    String type = staffInfo[0];
    String staffBarcode = staffInfo[1];
    double cost = paymentCalculator.calculate(type,
vehicleData.getVehicles().get(regNumber), timestamp);
    exitHandler.handleExit(timestamp, logger, regNumber, type, staffBarcode,
cost);
}

private boolean isFull() {
    return carPark.getFreeSpaces() == 0;
}
private boolean isEmpty() {
    return carPark.getOccupiedSpaces() == 0;
}

private String generateUniquePlate() throws FileNotFoundException {
    String regNumber;
    do {
        regNumber = numberPlateReader.read();
    } while (vehicleData.getVehicles().containsKey(regNumber));

    return regNumber;
}

private String checkStaff(String regNumber) throws FileNotFoundException {
    barcodeReader.setRegNumber(regNumber);
    String staffBarcode = barcodeReader.read();
    String type = (staffBarcode != null) ? "Staff" : "Customer";
    return (type + "|" + staffBarcode);
}
}

}

package CarPark.Models;

import CarPark.Utills.Observer.Observer;
import CarPark.Utills.Observer.Subject;

import java.util.ArrayList;
import java.util.List;

public class CarPark implements Subject {
    private int capacity;
    private int occupiedSpaces;
    private double pricePerHour;
    private String location;
    private final List<Observer> observers = new ArrayList<>();

    @Override    public void addObserver(Observer o) {
        observers.add(o);
    }

    @Override    public void removeObserver(Observer o) {
        observers.remove(o);
    }

    @Override    public void notifyObservers() {
        boolean full = occupiedSpaces >= capacity;
        int freeSpaces = capacity - occupiedSpaces;

```

```

        for (Observer o : observers) {
            o.update(full, freeSpaces);
        }
    }

    public int getFreeSpaces() {
        return capacity - occupiedSpaces;
    }

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

    public int getCapacity() {return capacity;}

    public void setOccupiedSpaces(int occupiedSpaces) {
        this.occupiedSpaces = occupiedSpaces;
        notifyObservers();
    }

    public int getOccupiedSpaces() {return occupiedSpaces;}

    public void setPricePerHour(double pricePerHour) {
        this.pricePerHour = pricePerHour;
    }

    public double getPricePerHour() {return pricePerHour;}

    public void setLocation(String location) {
        this.location = location;
    }

    public String getLocation() {return location;}
}

```

```

package CarPark.Models;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.*;

public class VehicleData {
    private HashMap<String, String> vehicles = new HashMap<>();
    private final Path VEHICLE_FILE = Paths.get("data", "vehicles.json");

    public HashMap<String,String> getVehicles() {
        return vehicles;
    }

    public boolean hasVehicles(String regNumber,String entryTime) {
        return vehicles.containsKey(regNumber);
    }

    public String getEntryTimeFor(String regNumber) {
        String entryTime = vehicles.get(regNumber.trim());
        return (entryTime != null) ? entryTime : "";
    }

    public VehicleData() {

```

```

        clearVehicleFile();
        loadAllVehicles(); // Loads nothing after deletion
    }

    public String peekRandomVehicle() {
        if (vehicles.isEmpty()) return null;
        List<String> keys = new ArrayList<>(vehicles.keySet());
        return keys.get(new Random().nextInt(keys.size()));
    }

    private void clearVehicleFile() {
        try {
            Files.createDirectories(VEHICLE_FILE.getParent());
        }

        // Always write valid empty JSON to start the file

        Files.write(VEHICLE_FILE, [].getBytes(), StandardOpenOption.CREATE,
StandardOpenOption.TRUNCATE_EXISTING);

        System.out.println("vehicles.json reset with empty array.");
    } catch (IOException e) {
        System.err.println("Failed to reset vehicles.json");
        e.printStackTrace();
    }
}

    public void addVehicle(String regNumber, String entryTime,String type,String
staffBarcode) {
        List<VehicleInfo> vehicleList = loadAllVehicles();
        vehicleList.removeIf(v -> v.getRegNumber().equals(regNumber));
        vehicleList.add(new VehicleInfo(regNumber, entryTime, "", type,
staffBarcode,"Parked"));

        vehicles.put(regNumber.trim(), entryTime.trim());
        saveAllVehicles(vehicleList);
    }

    public void markVehicleExited(String regNumber, String exitTime) {
        List<VehicleInfo> all = loadAllVehicles();
        for (VehicleInfo vehicle : all) {
            if (vehicle.getRegNumber().equals(regNumber)) {
                vehicle.setExitTime(exitTime);
                vehicle.setStatus("Exited");
                break;
            }
        }
        saveAllVehicles(all);
    }

    public void setVehicles(HashMap<String, String> vehicles) {
        this.vehicles = vehicles;
    }

    private void saveAllVehicles(List<VehicleInfo> list) {
        try {
            Files.createDirectories(VEHICLE_FILE.getParent());

            StringBuilder json = new StringBuilder("[\n");
            for(int i =0; i < list.size(); i++) {
                VehicleInfo vehicle = list.get(i);
                json.append(" {\n");
                json.append("     \"regNumber\":\n");
                json.append(vehicle.getRegNumber().trim()).append("\",\n");
                json.append("     \"entryTime\":\n");
                json.append(vehicle.getEntryTime().trim()).append("\",\n");
                json.append("     \"exitTime\":\n");
                json.append(vehicle.getExitTime().trim()).append("\",\n");
                json.append("     \"type\":\n");
                json.append(vehicle.getType().trim()).append("\",\n");
            }
        }
    }
}

```

```

        json.append("    \"staffBarcode\":\n");
        \").append(vehicle.getStaffBarcode().trim()).append("\",\n");
        json.append("    \"status\":\n");
        \").append(vehicle.getStatus().trim()).append("\",\n");
        json.append("  }");
        if(i< list.size() - 1) json.append(",");
        json.append("\n");
    }
    json.append("]");

    Files.write(VEHICLE_FILE, json.toString().getBytes(),
        StandardOpenOption.CREATE,
        StandardOpenOption.TRUNCATE_EXISTING);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private List<VehicleInfo> loadAllVehicles() {
    List<VehicleInfo> vehicleList = new ArrayList<>();
    vehicles.clear(); // Ensure fresh state
    try {
        if (!Files.exists(VEHICLE_FILE)) return vehicleList;

        String rawJson = new String(Files.readAllBytes(VEHICLE_FILE)).trim();
        if (rawJson.length() <= 2) return vehicleList;

        rawJson = rawJson.substring(1, rawJson.length() - 1).trim();
        String[] objects = rawJson.split("\\\\", \\s*\\\\");
        for (String object : objects) {
            String cleanObject = object.replaceAll("[\\\\{\\\\}\\\"\\\\]", "").trim();
            String[] fields = cleanObject.split(",");
            String regNumber = "", entryTime = "", exitTime = "", type = "",
            staffBarcode = "", status = "";
            for (String field : fields) {
                String[] keyValue = field.trim().split(":", 2);
                if (keyValue.length == 2) {
                    switch (keyValue[0].trim()) {
                        case "regNumber":
                            regNumber = keyValue[1].trim();
                            break;
                        case "entryTime":
                            entryTime = keyValue[1].trim();
                            break;
                        case "exitTime":
                            exitTime = keyValue[1].trim();
                            break;
                        case "type":
                            type = keyValue[1].trim();
                            break;
                        case "staffBarcode":
                            staffBarcode = keyValue[1].trim();
                            break;
                        case "status":
                            status = keyValue[1].trim();
                            break;
                    }
                }
            }
            if (!regNumber.isEmpty()) {
                vehicleList.add(new VehicleInfo(regNumber, entryTime, exitTime,
                type, staffBarcode, status));
                if ("Parked".equals(status)) {
                    vehicles.put(regNumber, entryTime);
                }
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

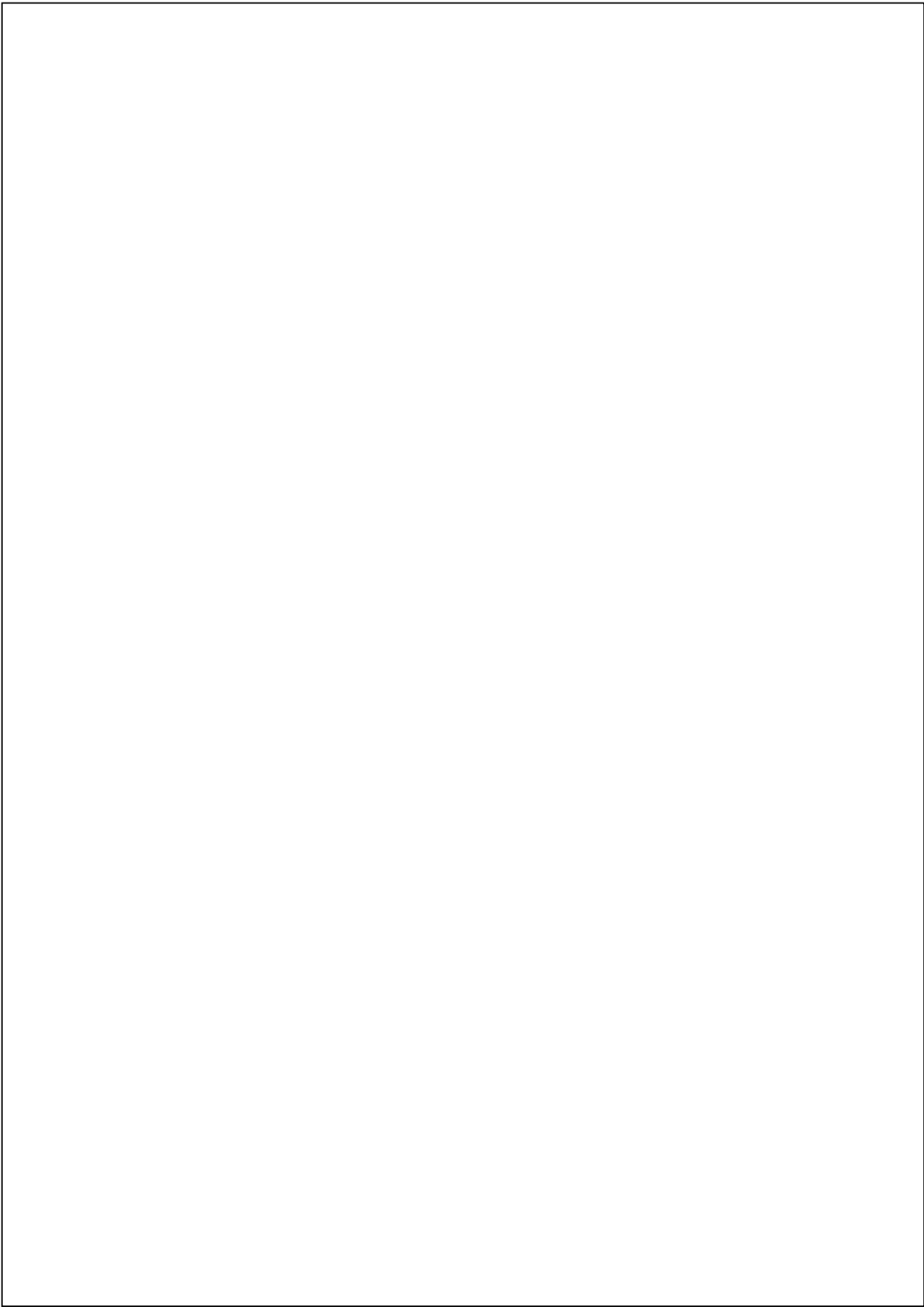
```
    }  
    return vehicleList;  
}  
}
```

```
package CarPark.Models;  
  
public class VehicleInfo {  
    private String regNumber;  
  
    private String entryTime;  
    private String exitTime;  
    private String type;  
    private String staffBarcode;  
    private String status;  
  
    public VehicleInfo(String regNumber, String entryTime, String exitTime, String  
type, String staffBarcode, String status) {  
        this.regNumber = regNumber;  
        this.entryTime = entryTime;  
        this.exitTime = exitTime;  
        this.type = type;  
        this.staffBarcode = staffBarcode;  
        this.status = status;  
    }  
  
    public String getRegNumber() {  
        return regNumber;  
    }  
  
    public String getEntryTime() {  
        return entryTime;  
    }  
  
    public String getExitTime() {  
        return exitTime;  
    }  
  
    public String getType() {  
        return type;  
    }  
  
    public String getStaffBarcode() {  
        return staffBarcode;  
    }  
  
    public String getStatus() {  
        return status;  
    }  
  
    public void setRegNumber(String regNumber) {  
        this.regNumber = regNumber;  
    }  
  
    public void setEntryTime(String entryTime) {  
        this.entryTime = entryTime;  
    }  
}
```

```
public void setExitTime(String exitTime) {  
    this.exitTime = exitTime;  
}  
  
public void setType(String type) {  
    this.type = type;  
}  
  
public void setStaffBarcode(String staffBarcode) {  
    this.staffBarcode = staffBarcode;  
}  
  
public void setStatus(String status) {  
    this.status = status;  
}  
}
```

References

- [1] Z. Rashidi, "Properties of Relationships among objects in Object-Oriented Software Design," 2015, *Ithaca*. [Online]. Available: <https://www.proquest.com/working-papers/properties-relationships-among-objects-object/docview/2083854869/se-2?accountid=130472>
- [2] D. C. Halbert and P. D. O'Brien, "Using Types and Inheritance in Object-Oriented Programming," *IEEE Softw*, vol. 4, no. 5, pp. 71–79, 1987, doi: <https://doi.org/10.1109/MS.1987.231776>.
- [3] H. Hourani, H. Wasmi, and T. Alrawashdeh, "A Code Complexity Model of Object Oriented Programming (OOP)," in *JEEIT*, IEEE, 2019, pp. 560–564. doi: 10.1109/JEEIT.2019.8717448.
- [4] H. Ali, "Exploring Polymorphism: Flexibility and Code Reusability in Object-Oriented Programming," *Passer journal of basic and applied sciences (Online)*, vol. 6, no. Special Issue, 2024, doi: 10.24271/psr.2024.189667.
- [5] T. Larsson, "System and method for reducing coupling in an object-oriented programming environment," 1999.
- [6] J. Garzas and M. Piattini, *Object-Oriented Design Knowledge: Principles, Heuristics and Best Practices*, 1st ed. Hershey: Idea Group, Inc, 2006. doi: 10.4018/978-1-59140-896-3.
- [7] L. JiCheng, W. YuHua, W. YaBo, and X. Su, "The implementation of Observer pattern by aspect based on properties file," in *2009 4th International Conference on Computer Science & Education*, 2009, pp. 908–912. doi: 10.1109/ICCSE.2009.5228245.
- [8] E. Corry, "Optimistic stack allocation for java-like languages," New York, NY, USA: ACM, 2006, pp. 162–173. doi: 10.1145/1133956.1133978.
- [9] A. S. Fong and R. C. L. Li, "Dynamic memory allocation/deallocation behavior in Java programs," in *TENCON*, IEEE, 2002, pp. 314–317 vol.1. doi: 10.1109/TENCON.2002.1181277.
- [10] T. Kim and H. Kim, "LWMalloc: A Lightweight Dynamic Memory Allocator for Resource-Constrained Environments," *IEEE Internet Things J*, p. 1, 2025, doi: 10.1109/JIOT.2025.3541247.
- [11] S. Haines, "JUnit 5 tutorial, part 1: Unit testing with JUnit 5, Mockito, and Hamcrest: Set up your first Maven project and start writing robust unit tests with JUnit 5, Hamcrest, and Mockito," *JavaWorld*, May 2020, [Online]. Available: <https://www.proquest.com/trade-journals/junit-5-tutorial-part-1-unit-testing-with-mockito/docview/2398373081/se-2?accountid=130472>



FINAL GRADE

81 /100

GENERAL COMMENTS

Engagement with Literature (5 marks): 3/5

Range & credibility of sources 1/2

Correct in-text citations & IEEE formatting 1/2

Evidence of critical reading 1/1

Knowledge & Understanding (20 marks): 15/20

Best practice analysis (e.g. OO principles) 5/6

Static vs dynamic memory explanation 5/7

Data-structure justification 5/7

Cognitive & Intellectual Skills (20 marks): 16/20

Critical evaluation of designs & literature 7/10

Justification of hypotheses/decisions 4/5

Synthesis of complex issues 5/5

Practical Application Skills (50 marks): 43/50

Class design & file organisation 7/10

Implementation of core simulation 8/10

Inheritance & IDReader subclasses 5/5

Dynamic data-structure coding 5/5

Use of OO concepts (abstract classes,
polymorphism,
generics,

inheritance,

interfaces) 8/ 10

Memory-based storage structures 5/5

Unit tests & harness 5/5

~~~~~

### **Transferable Skills (5 marks):4/5**

Report structure & clarity 1/2

Academic writing & organisation 1/1

User guide clarity 1/1

Video presentation quality 1/1

Comment:

Great effort, Ryan, A handful of minor report things to consider going forward!

---

PAGE 1

---

PAGE 2

---

PAGE 3



#### **Comment 1**

Typically in the IEEE referencing style we'll put the reference at the end of the sentence



#### **Comment 2**

images should have a caption along with a table of figures for easier navigation

---

PAGE 4



#### **Comment 3**

the reference should have been behind the first full stop

---

PAGE 5

---

PAGE 6



#### **Comment 4**

As a title this should be a capital P

PAGE 7

---

PAGE 8

---

PAGE 9

---

PAGE 10

---

PAGE 11

---

PAGE 12

---

PAGE 13

---

PAGE 14

---

PAGE 15

---



### Comment 5

Love it

PAGE 16

---



### Comment 6

Good stuff

PAGE 17

---



### Comment 7

References before appendix!

PAGE 18

---

PAGE 19

---

PAGE 20

---

PAGE 21

---

PAGE 22

---

PAGE 23

---

PAGE 24

---

PAGE 25

---

PAGE 26

---

PAGE 27

---

PAGE 28

---

PAGE 29

---

PAGE 30

---

