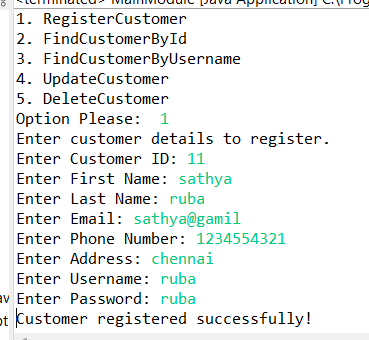
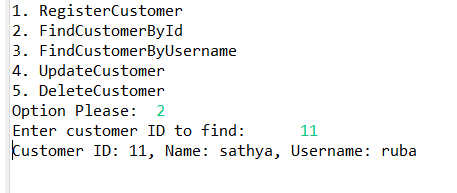
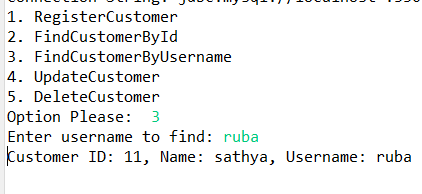
**1.Register Customer**



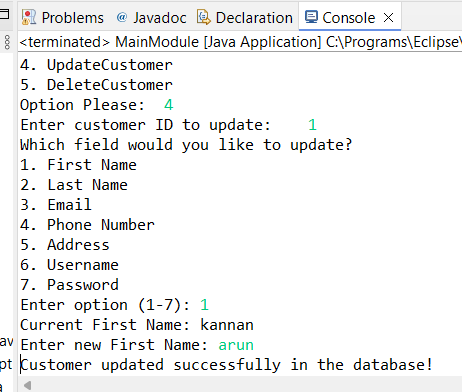
**2)find customer by id**



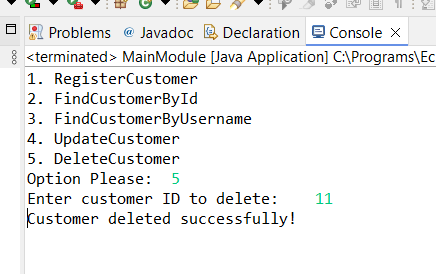
**3)find customer by username**



**4)Update customer**

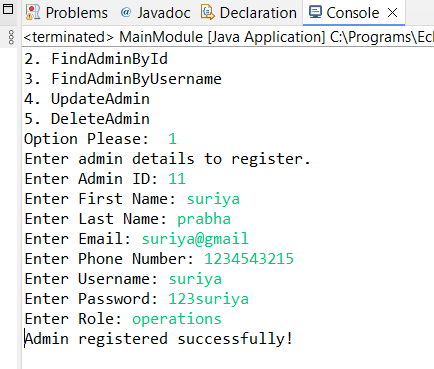


**5) delete**

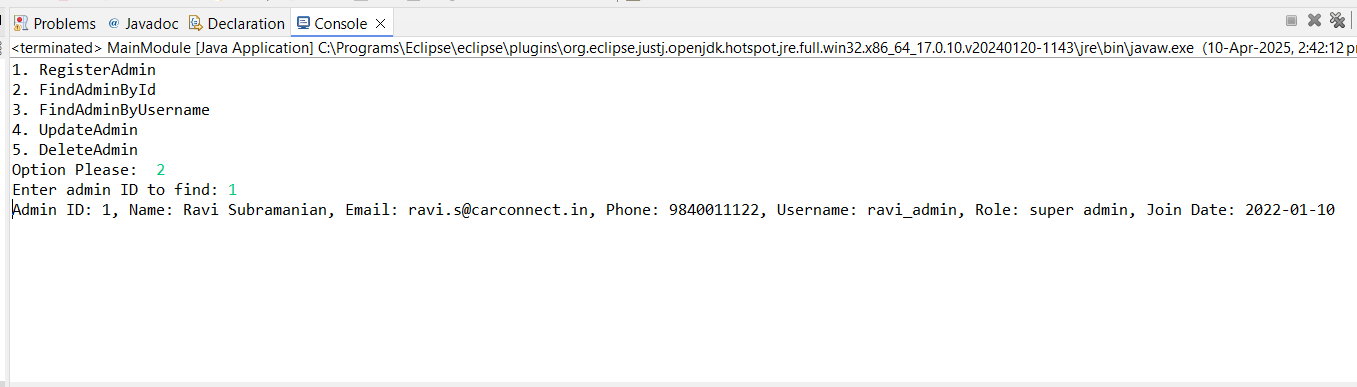


**Admin**

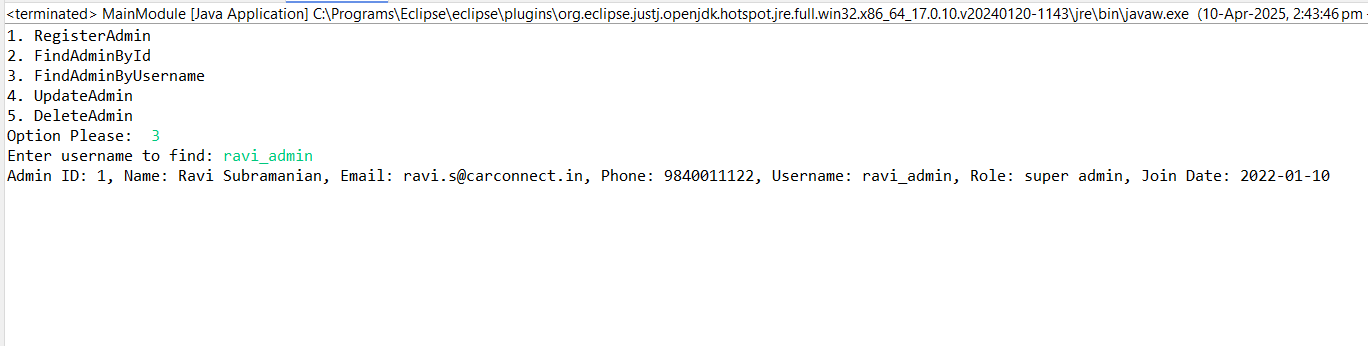
**1)Register admin**



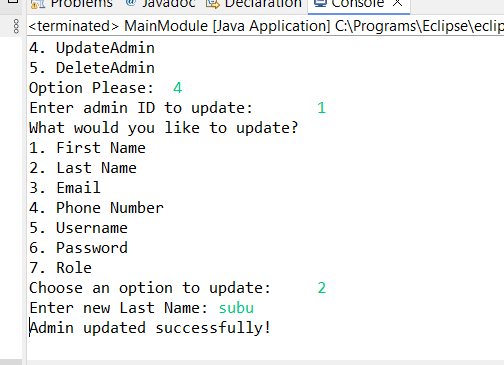
**2) find admin by id**



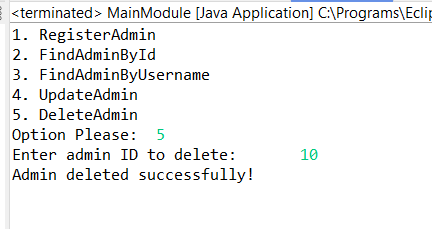
**3)find admin by username**



**4)Update admin**

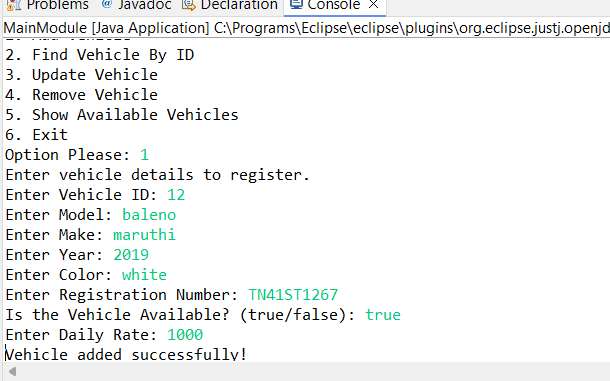


**5)delete admin**

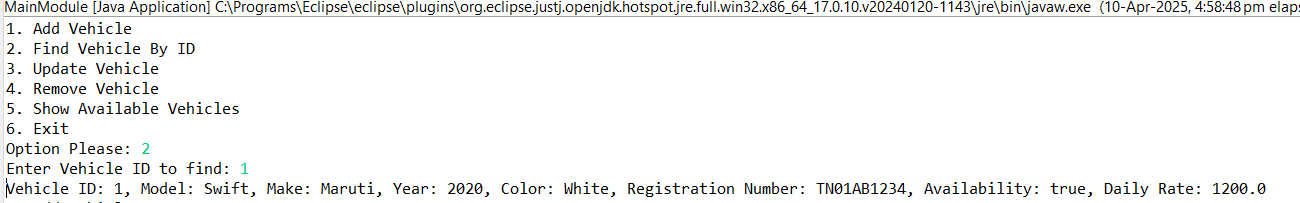


**Vehicle**

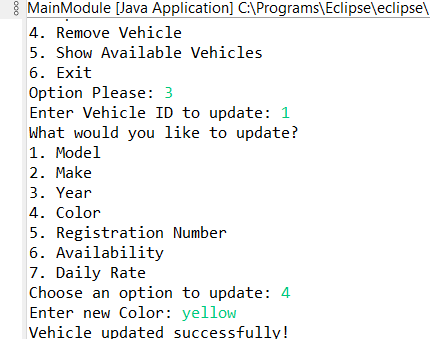
**1) register vehicle**



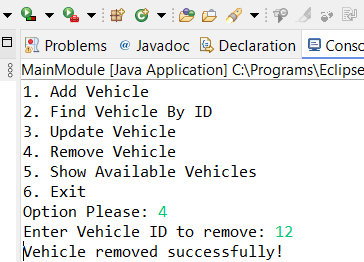
**2)find vehicle by id**



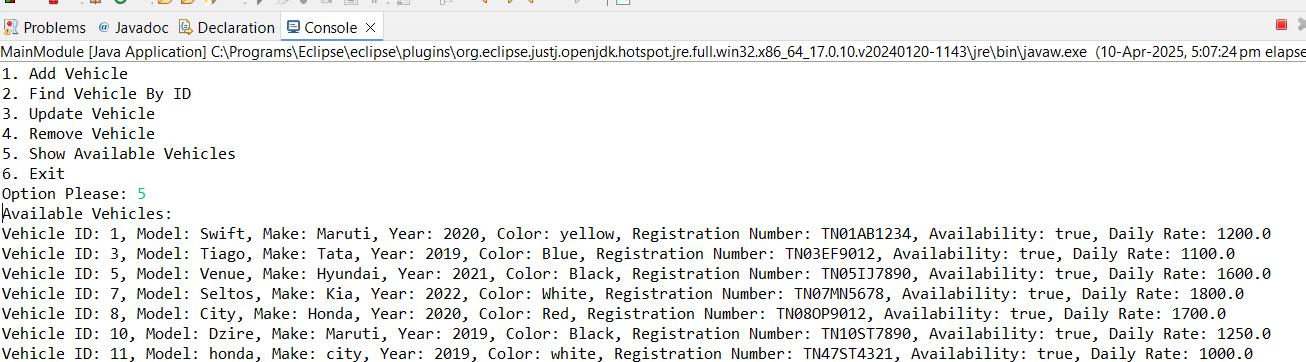
**3)update vehicle**



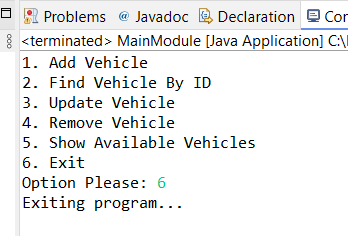
**4)remove vehicles**



**5)show all available vehicles**

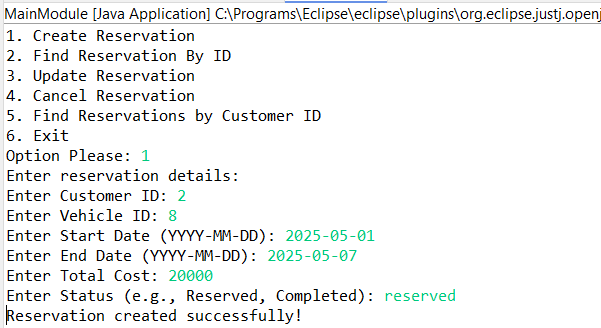


**6)exit**

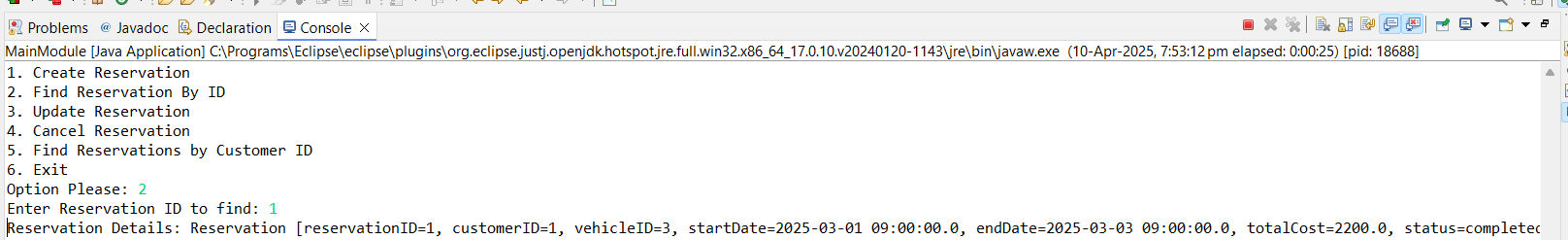


**reservation**

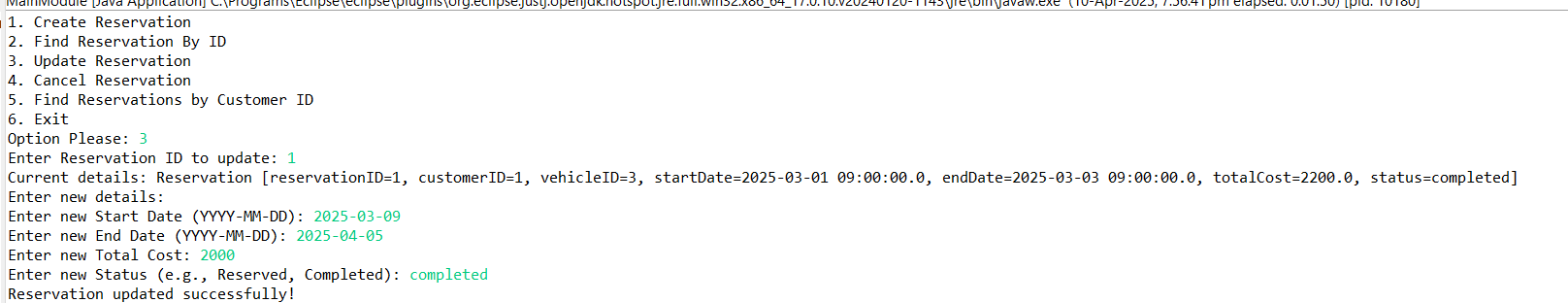
**1)create reservation**



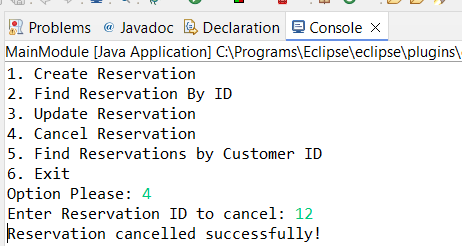
**2) find reservation by reservation id**



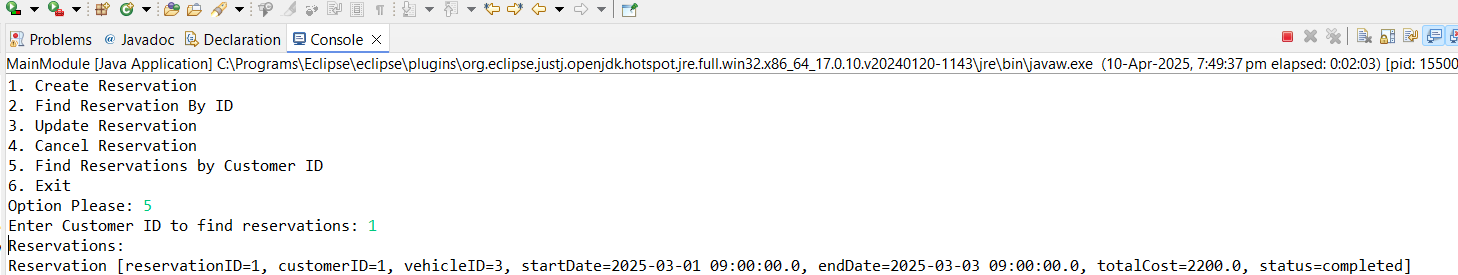
**3)update reservation**



**4)delete reservation**



**5)find reservation by customer id**



**ENTITY PACKAGE**

Admin

**package** entity;

**import** java.util.Date;

**public** **class** Admin {

**private** **int** adminID;

**private** String firstName;

**private** String lastName;

**private** String email;

**private** String phoneNumber;

**private** String username;

**private** String password;

**private** String role;

**private** Date joinDate;

**public** Admin()

{

}

**public** Admin(**int** adminID, String firstName, String lastName, String email, String phoneNumber, String username,

String password, String role, Date joinDate) {

**super**();

**this**.adminID = adminID;

**this**.firstName = firstName;

**this**.lastName = lastName;

**this**.email = email;

**this**.phoneNumber = phoneNumber;

**this**.username = username;

**this**.password = password;

**this**.role = role;

**this**.joinDate = joinDate;

}

**public** **int** getAdminID() {

**return** adminID;

}

**public** **void** setAdminID(**int** adminID) {

**this**.adminID = adminID;

}

**public** String getFirstName() {

**return** firstName;

}

**public** **void** setFirstName(String firstName) {

**this**.firstName = firstName;

}

**public** String getLastName() {

**return** lastName;

}

**public** **void** setLastName(String lastName) {

**this**.lastName = lastName;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getPhoneNumber() {

**return** phoneNumber;

}

**public** **void** setPhoneNumber(String phoneNumber) {

**this**.phoneNumber = phoneNumber;

}

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** String getRole() {

**return** role;

}

**public** **void** setRole(String role) {

**this**.role = role;

}

**public** Date getJoinDate() {

**return** joinDate;

}

**public** **void** setJoinDate(Date joinDate) {

**this**.joinDate = joinDate;

}

**public** **boolean** authenticate(String inputPassword) {

**return** **this**.password != **null** && **this**.password.equals(inputPassword);

}

@Override

**public** String toString() {

**return** "Admin ID: " + adminID + ", Name: " + firstName + " " + lastName +

", Email: " + email + ", Phone: " + phoneNumber + ", Username: " + username +

", Role: " + role + ", Join Date: " + joinDate;

}

}

**Customer**

**package** entity;

**import** java.util.Date;

**public** **class** Customer {

**private** **int** customerID;

**private** String firstName;

**private** String lastName;

**private** String email;

**private** String phoneNumber;

**private** String address;

**private** String username;

**private** String password;

**private** Date registrationDate;

**public** Customer()

{

}

**public** Customer(**int** customerID, String firstName, String lastName, String email, String phoneNumber,

String address, String username, String password, Date registrationDate) {

**super**();

**this**.customerID = customerID;

**this**.firstName = firstName;

**this**.lastName = lastName;

**this**.email = email;

**this**.phoneNumber = phoneNumber;

**this**.address = address;

**this**.username = username;

**this**.password = password;

**this**.registrationDate = registrationDate;

}

**public** **int** getCustomerID() {

**return** customerID;

}

**public** **void** setCustomerID(**int** customerID) {

**this**.customerID = customerID;

}

**public** String getFirstName() {

**return** firstName;

}

**public** **void** setFirstName(String firstName) {

**this**.firstName = firstName;

}

**public** String getLastName() {

**return** lastName;

}

**public** **void** setLastName(String lastName) {

**this**.lastName = lastName;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getPhoneNumber() {

**return** phoneNumber;

}

**public** **void** setPhoneNumber(String phoneNumber) {

**this**.phoneNumber = phoneNumber;

}

**public** String getAddress() {

**return** address;

}

**public** **void** setAddress(String address) {

**this**.address = address;

}

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** Date getRegistrationDate() {

**return** registrationDate;

}

**public** **void** setRegistrationDate(Date registrationDate) {

**this**.registrationDate = registrationDate;

}

**public** **boolean** authenticate(String inputPassword) {

**return** **this**.password.equals(inputPassword);

}

@Override

**public** String toString() {

**return** "Customer ID: " + customerID + ", Name: " + firstName + ", Username: " + username;

}

}

**Reservation**

**package** entity;

**import** java.util.Date;

**public** **class** Reservation {

**private** **int** reservationID;

**private** **int** customerID;

**private** **int** vehicleID;

**private** Date startDate;

**private** Date endDate;

**private** **double** totalCost;

**private** String status;

**public** Reservation()

{

}

**public** Reservation(**int** reservationID, **int** customerID, **int** vehicleID, Date startDate, Date endDate,

**double** totalCost, String status) {

**super**();

**this**.reservationID = reservationID;

**this**.customerID = customerID;

**this**.vehicleID = vehicleID;

**this**.startDate = startDate;

**this**.endDate = endDate;

**this**.totalCost = totalCost;

**this**.status = status;

}

**public** **int** getReservationID() {

**return** reservationID;

}

**public** **void** setReservationID(**int** reservationID) {

**this**.reservationID = reservationID;

}

**public** **int** getCustomerID() {

**return** customerID;

}

**public** **void** setCustomerID(**int** customerID) {

**this**.customerID = customerID;

}

**public** **int** getVehicleID() {

**return** vehicleID;

}

**public** **void** setVehicleID(**int** vehicleID) {

**this**.vehicleID = vehicleID;

}

**public** Date getStartDate() {

**return** startDate;

}

**public** **void** setStartDate(Date startDate) {

**this**.startDate = startDate;

}

**public** Date getEndDate() {

**return** endDate;

}

**public** **void** setEndDate(Date endDate) {

**this**.endDate = endDate;

}

**public** **double** getTotalCost() {

**return** totalCost;

}

**public** **void** setTotalCost(**double** totalCost) {

**this**.totalCost = totalCost;

}

**public** String getStatus() {

**return** status;

}

**public** **void** setStatus(String status) {

**this**.status = status;

}

@Override

**public** String toString() {

**return** "Reservation [reservationID=" + reservationID +

", customerID=" + customerID +

", vehicleID=" + vehicleID +

", startDate=" + startDate +

", endDate=" + endDate +

", totalCost=" + totalCost +

", status=" + status + "]";

}

**public** **double** calculateTotalCost(**double** dailyRate) {

**long** diffInMillies = Math.*abs*(endDate.getTime() - startDate.getTime());

**long** days = (diffInMillies / (1000 \* 60 \* 60 \* 24)) + 1; // +1 to count the start day

**return** dailyRate \* days;

}

}

**Vehicle**

**package** entity;

**public** **class** Vehicle {

**private** **int** vehicleID;

**private** String model;

**private** String make;

**private** **int** year;

**private** String color;

**private** String registrationNumber;

**private** **boolean** availability;

**private** **double** dailyRate;

**public** Vehicle()

{

}

**public** Vehicle(**int** vehicleID, String model, String make, **int** year, String color, String registrationNumber,

**boolean** availability, **double** dailyRate) {

**super**();

**this**.vehicleID = vehicleID;

**this**.model = model;

**this**.make = make;

**this**.year = year;

**this**.color = color;

**this**.registrationNumber = registrationNumber;

**this**.availability = availability;

**this**.dailyRate = dailyRate;

}

**public** **int** getVehicleID() {

**return** vehicleID;

}

**public** **void** setVehicleID(**int** vehicleID) {

**this**.vehicleID = vehicleID;

}

**public** String getModel() {

**return** model;

}

**public** **void** setModel(String model) {

**this**.model = model;

}

**public** String getMake() {

**return** make;

}

**public** **void** setMake(String make) {

**this**.make = make;

}

**public** **int** getYear() {

**return** year;

}

**public** **void** setYear(**int** year) {

**this**.year = year;

}

**public** String getColor() {

**return** color;

}

**public** **void** setColor(String color) {

**this**.color = color;

}

**public** String getRegistrationNumber() {

**return** registrationNumber;

}

**public** **void** setRegistrationNumber(String registrationNumber) {

**this**.registrationNumber = registrationNumber;

}

**public** **boolean** isAvailability() {

**return** availability;

}

**public** **void** setAvailability(**boolean** availability) {

**this**.availability = availability;

}

**public** **double** getDailyRate() {

**return** dailyRate;

}

**public** **void** setDailyRate(**double** dailyRate) {

**this**.dailyRate = dailyRate;

}

@Override

**public** String toString() {

**return** "Vehicle ID: " + vehicleID + ", Model: " + model + ", Make: " + make +

", Year: " + year + ", Color: " + color + ", Registration Number: " + registrationNumber +

", Availability: " + availability + ", Daily Rate: " + dailyRate;

}

}

**UTIL**

**DBConnUtil**

package util;

import java.io.IOException;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import exception.DatabaseConnectionException;

public class DBConnUtil {

private static final String fileName = "db.properties";

public static Connection getDbConnection() throws DatabaseConnectionException {

Connection con = null;

String connString = null;

try {

connString = DBPropertyUtil.getConnectionString(fileName);

} catch (IOException e) {

throw new DatabaseConnectionException("Connection String Creation Failed", e);

}

if (connString != null) {

//System.out.println("Connection String: " + connString);

try {

con = DriverManager.getConnection(connString);

} catch (SQLException e) {

throw new DatabaseConnectionException("Error While Establishing DBConnection........", e);

}

}

return con;

}

}

**DBPropertyUtil**

package util;

import java.io.FileInputStream;

import java.io.IOException;

import java.util.Properties;

public class DBPropertyUtil {

public static String getConnectionString(String fileName)throws IOException {

//fileName="db.properties"

String connStr=null;

Properties props=new Properties();

FileInputStream fis=new FileInputStream(fileName);

props.load(fis);

String user=props.getProperty("user");

String password=props.getProperty("password");

String protocol=props.getProperty("protocol");

String system=props.getProperty("system");

String database=props.getProperty("database");

String port=props.getProperty("port");

connStr = protocol + "//" + system + ":" + port + "/" + database + "?user=" + user + "&password=" + password;

return connStr;

}

}

**Exception**

**AdminNotFoundException**

**package** exception;

**public** **class** AdminNotFoundException **extends** Exception {

**public** AdminNotFoundException(String message) {

**super**(message);

}

**public** AdminNotFoundException(String message, Throwable cause) {

**super**(message, cause);

}

}

**AuthenticationException**

**package** exception;

**public** **class** AuthenticationException **extends** Exception {

**public** AuthenticationException(String message) {

**super**(message);

}

**public** AuthenticationException(String message, Throwable cause) {

**super**(message, cause);

}

}

**DatabaseConnectionException**

package exception;

public class DatabaseConnectionException extends Exception {

public DatabaseConnectionException(String message) {

super(message);

}

public DatabaseConnectionException(String message, Throwable cause) {

super(message, cause);

}

}

**InvalidInputException**

**package** exception;

**public** **class** InvalidInputException **extends** Exception {

**public** InvalidInputException(String message) {

**super**(message);

}

**public** InvalidInputException(String message, Throwable cause) {

**super**(message, cause);

}

}

**ReservationException**

package exception;

public class ReservationException extends Exception {

public ReservationException(String message) {

super(message);

}

public ReservationException(String message, Throwable cause) {

super(message, cause);

}

}

**VehicleNotFoundException**

**package** exception;

**public** **class** VehicleNotFoundException **extends** Exception{

**public** VehicleNotFoundException(String message) {

**super**(message);

}

}

**DAO**

**ICustomerService**

package dao;

import entity.Customer;

import exception.\*;

public interface ICustomerService {

Customer getCustomerById(int customerId) throws InvalidInputException;

Customer getCustomerByUsername(String username) throws AuthenticationException;

boolean registerCustomer(Customer customer) throws InvalidInputException, DatabaseConnectionException;

boolean updateCustomer(Customer customer) throws InvalidInputException, DatabaseConnectionException;

boolean deleteCustomer(int customerId) throws InvalidInputException, DatabaseConnectionException;

}

**CustomerService**

package dao;

import entity.Customer;

import exception.\*;

import util.DBConnUtil;

import java.sql.\*;

//import java.util.Date;

public class CustomerService implements ICustomerService {

private Connection con;

public CustomerService() throws DatabaseConnectionException {

super();

con = DBConnUtil.getDbConnection();

}

@Override

public Customer getCustomerById(int customerId) throws InvalidInputException {

String sql = "SELECT \* FROM Customer WHERE CustomerID = ?";

Customer customer = null;

try (PreparedStatement pstmt = con.prepareStatement(sql)) {

pstmt.setInt(1, customerId);

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

customer = new Customer(

rs.getInt("CustomerID"),

rs.getString("FirstName"),

rs.getString("LastName"),

rs.getString("Email"),

rs.getString("PhoneNumber"),

rs.getString("Address"),

rs.getString("Username"),

rs.getString("Password"),

rs.getDate("RegistrationDate")

);

} else {

throw new InvalidInputException("Customer with ID " + customerId + " not found.");

}

} catch (SQLException e) {

throw new InvalidInputException("Error fetching customer data: " + e.getMessage(), e);

}

return customer;

}

@Override

public Customer getCustomerByUsername(String username) throws AuthenticationException {

String sql = "SELECT \* FROM Customer WHERE Username = ?";

Customer customer = null;

try (PreparedStatement pstmt = con.prepareStatement(sql)) {

pstmt.setString(1, username);

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

customer = new Customer(

rs.getInt("CustomerID"),

rs.getString("FirstName"),

rs.getString("LastName"),

rs.getString("Email"),

rs.getString("PhoneNumber"),

rs.getString("Address"),

rs.getString("Username"),

rs.getString("Password"),

rs.getDate("RegistrationDate")

);

} else {

throw new AuthenticationException("Username not found.");

}

} catch (SQLException e) {

throw new AuthenticationException("Database error: " + e.getMessage(), e);

}

return customer;

}

@Override

public boolean registerCustomer(Customer customer) throws InvalidInputException, DatabaseConnectionException {

String sql = "INSERT INTO Customer (CustomerID, FirstName, LastName, Email, PhoneNumber, Address, Username, Password, RegistrationDate) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)";

try (PreparedStatement pstmt = con.prepareStatement(sql)) {

pstmt.setInt(1, customer.getCustomerID());

pstmt.setString(2, customer.getFirstName());

pstmt.setString(3, customer.getLastName());

pstmt.setString(4, customer.getEmail());

pstmt.setString(5, customer.getPhoneNumber());

pstmt.setString(6, customer.getAddress());

pstmt.setString(7, customer.getUsername());

pstmt.setString(8, customer.getPassword());

pstmt.setDate(9, new java.sql.Date(customer.getRegistrationDate().getTime()));

pstmt.executeUpdate();

return true;

} catch (SQLException e) {

throw new DatabaseConnectionException("Error inserting customer: " + e.getMessage(), e);

}

}

@Override

public boolean updateCustomer(Customer customer) throws InvalidInputException, DatabaseConnectionException {

String sql = "UPDATE Customer SET FirstName=?, LastName=?, Email=?, PhoneNumber=?, Address=?, Username=?, Password=? WHERE CustomerID=?";

try (PreparedStatement pstmt = con.prepareStatement(sql)) {

pstmt.setString(1, customer.getFirstName());

pstmt.setString(2, customer.getLastName());

pstmt.setString(3, customer.getEmail());

pstmt.setString(4, customer.getPhoneNumber());

pstmt.setString(5, customer.getAddress());

pstmt.setString(6, customer.getUsername());

pstmt.setString(7, customer.getPassword());

pstmt.setInt(8, customer.getCustomerID());

pstmt.executeUpdate();

return true;

} catch (SQLException e) {

throw new DatabaseConnectionException("Error updating customer: " + e.getMessage(), e);

}

}

@Override

public boolean deleteCustomer(int customerId) throws InvalidInputException, DatabaseConnectionException {

String sql = "DELETE FROM Customer WHERE CustomerID = ?";

try (PreparedStatement pstmt = con.prepareStatement(sql)) {

pstmt.setInt(1, customerId);

int rowsAffected = pstmt.executeUpdate();

if (rowsAffected == 0) {

throw new InvalidInputException("Customer with ID " + customerId + " not found.");

}

return true;

} catch (SQLException e) {

throw new DatabaseConnectionException("Error deleting customer: " + e.getMessage(), e);

}

}

}

**IAdminService**

package dao;

import entity.Admin;

import exception.AdminNotFoundException;

import exception.InvalidInputException;

public interface IAdminService {

Admin getAdminById(int adminId) throws AdminNotFoundException;

Admin getAdminByUsername(String username) throws AdminNotFoundException;

boolean registerAdmin(Admin adminData) throws InvalidInputException;

boolean updateAdmin(Admin adminData) throws AdminNotFoundException;

boolean deleteAdmin(int adminId) throws AdminNotFoundException;

}

**AdminService**

package dao;

import entity.Admin;

import exception.AdminNotFoundException;

import exception.InvalidInputException;

import exception.DatabaseConnectionException;

import util.DBConnUtil;

import java.sql.\*;

public class AdminService implements IAdminService {

private Connection con;

public AdminService() throws DatabaseConnectionException {

con = DBConnUtil.getDbConnection();

}

@Override

public Admin getAdminById(int adminId) throws AdminNotFoundException {

String sql = "SELECT \* FROM Admin WHERE AdminID = ?";

try (PreparedStatement pstmt = con.prepareStatement(sql)) {

pstmt.setInt(1, adminId);

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

return extractAdminFromResultSet(rs);

} else {

throw new AdminNotFoundException("Admin with ID " + adminId + " not found.");

}

} catch (SQLException e) {

throw new AdminNotFoundException("Error fetching admin by ID: " + e.getMessage(), e);

}

}

@Override

public Admin getAdminByUsername(String username) throws AdminNotFoundException {

String sql = "SELECT \* FROM Admin WHERE Username = ?";

try (PreparedStatement pstmt = con.prepareStatement(sql)) {

pstmt.setString(1, username);

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

return extractAdminFromResultSet(rs);

} else {

throw new AdminNotFoundException("Admin with username '" + username + "' not found.");

}

} catch (SQLException e) {

throw new AdminNotFoundException("Error fetching admin by username: " + e.getMessage(), e);

}

}

@Override

public boolean registerAdmin(Admin adminData) throws InvalidInputException {

String sql = "INSERT INTO Admin (AdminID, FirstName, LastName, Email, PhoneNumber, Username, Password, Role) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";

try (PreparedStatement pstmt = con.prepareStatement(sql)) {

pstmt.setInt(1, adminData.getAdminID());

pstmt.setString(2, adminData.getFirstName());

pstmt.setString(3, adminData.getLastName());

pstmt.setString(4, adminData.getEmail());

pstmt.setString(5, adminData.getPhoneNumber());

pstmt.setString(6, adminData.getUsername());

pstmt.setString(7, adminData.getPassword());

pstmt.setString(8, adminData.getRole());

return pstmt.executeUpdate() > 0;

} catch (SQLException e) {

throw new InvalidInputException("Error registering admin: " + e.getMessage(), e);

}

}

@Override

public boolean updateAdmin(Admin adminData) throws AdminNotFoundException {

String sql = "UPDATE Admin SET FirstName=?, LastName=?, Email=?, PhoneNumber=?, Username=?, Password=?, Role=? WHERE AdminID=?";

try (PreparedStatement pstmt = con.prepareStatement(sql)) {

pstmt.setString(1, adminData.getFirstName());

pstmt.setString(2, adminData.getLastName());

pstmt.setString(3, adminData.getEmail());

pstmt.setString(4, adminData.getPhoneNumber());

pstmt.setString(5, adminData.getUsername());

pstmt.setString(6, adminData.getPassword());

pstmt.setString(7, adminData.getRole());

pstmt.setInt(8, adminData.getAdminID());

int rowsUpdated = pstmt.executeUpdate();

if (rowsUpdated == 0) {

throw new AdminNotFoundException("Admin with ID " + adminData.getAdminID() + " not found for update.");

}

return true;

} catch (SQLException e) {

throw new AdminNotFoundException("Error updating admin: " + e.getMessage(), e);

}

}

@Override

public boolean deleteAdmin(int adminId) throws AdminNotFoundException {

String sql = "DELETE FROM Admin WHERE AdminID = ?";

try (PreparedStatement pstmt = con.prepareStatement(sql)) {

pstmt.setInt(1, adminId);

int rowsDeleted = pstmt.executeUpdate();

if (rowsDeleted == 0) {

throw new AdminNotFoundException("Admin with ID " + adminId + " not found for deletion.");

}

return true;

} catch (SQLException e) {

throw new AdminNotFoundException("Error deleting admin: " + e.getMessage(), e);

}

}

private Admin extractAdminFromResultSet(ResultSet rs) throws SQLException {

return new Admin(

rs.getInt("AdminID"),

rs.getString("FirstName"),

rs.getString("LastName"),

rs.getString("Email"),

rs.getString("PhoneNumber"),

rs.getString("Username"),

rs.getString("Password"),

rs.getString("Role"),

rs.getDate("JoinDate")

);

}

}

**IVehicleService**

**package** dao;

**import** java.util.HashMap;

**import** entity.Vehicle;

**import** exception.\*;

**public** **interface** IVehicleService {

Vehicle getVehicleById(**int** vehicleId) **throws** VehicleNotFoundException;

HashMap<Integer, Vehicle> getAvailableVehicles();

**boolean** addVehicle(Vehicle vehicle) **throws** InvalidInputException;

**boolean** updateVehicle(Vehicle vehicle) **throws** VehicleNotFoundException;

**boolean** removeVehicle(**int** vehicleId) **throws** VehicleNotFoundException;

}

**VehicleService**

package dao;

import java.sql.\*;

import java.util.HashMap;

import entity.Vehicle;

import exception.DatabaseConnectionException;

import exception.VehicleNotFoundException;

import exception.InvalidInputException;

import util.DBConnUtil;

public class VehicleService implements IVehicleService {

private Connection con;

public VehicleService() throws DatabaseConnectionException {

super();

con = DBConnUtil.getDbConnection();

}

@Override

public Vehicle getVehicleById(int vehicleId) throws VehicleNotFoundException {

Vehicle vehicle = null;

try {

PreparedStatement pstmt = con.prepareStatement("SELECT \* FROM Vehicle WHERE VehicleID = ?");

pstmt.setInt(1, vehicleId);

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

vehicle = new Vehicle(

rs.getInt("VehicleID"),

rs.getString("Model"),

rs.getString("Make"),

rs.getInt("Year"),

rs.getString("Color"),

rs.getString("RegistrationNumber"),

rs.getBoolean("Availability"),

rs.getDouble("DailyRate")

);

} else {

throw new VehicleNotFoundException("Vehicle with ID " + vehicleId + " not found.");

}

} catch (SQLException e) {

System.out.println("Error retrieving vehicle by ID");

e.printStackTrace();

}

return vehicle;

}

@Override

public HashMap<Integer, Vehicle> getAvailableVehicles() {

HashMap<Integer, Vehicle> availableVehicles = new HashMap<>();

try {

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM Vehicle WHERE Availability = true");

while (rs.next()) {

Vehicle vehicle = new Vehicle(

rs.getInt("VehicleID"),

rs.getString("Model"),

rs.getString("Make"),

rs.getInt("Year"),

rs.getString("Color"),

rs.getString("RegistrationNumber"),

rs.getBoolean("Availability"),

rs.getDouble("DailyRate")

);

availableVehicles.put(vehicle.getVehicleID(), vehicle);

}

} catch (SQLException e) {

System.out.println("Error retrieving available Vehicle");

e.printStackTrace();

}

return availableVehicles;

}

@Override

public boolean addVehicle(Vehicle vehicle) throws InvalidInputException {

if (vehicle.getModel().isEmpty() || vehicle.getMake().isEmpty()) {

throw new InvalidInputException("Model and Make cannot be empty.");

}

if (vehicle.getYear() <= 0) {

throw new InvalidInputException("Year must be a positive number.");

}

if (vehicle.getDailyRate() <= 0) {

throw new InvalidInputException("Daily Rate must be a positive number.");

}

boolean flag = false;

try {

PreparedStatement pstmt = con.prepareStatement(

"INSERT INTO Vehicle (Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate) VALUES (?, ?, ?, ?, ?, ?, ?)"

);

pstmt.setString(1, vehicle.getModel());

pstmt.setString(2, vehicle.getMake());

pstmt.setInt(3, vehicle.getYear());

pstmt.setString(4, vehicle.getColor());

pstmt.setString(5, vehicle.getRegistrationNumber());

pstmt.setBoolean(6, vehicle.isAvailability());

pstmt.setDouble(7, vehicle.getDailyRate());

int rows = pstmt.executeUpdate();

if (rows > 0) {

flag = true;

}

} catch (SQLException e) {

System.out.println("Error while adding vehicle");

e.printStackTrace();

}

return flag;

}

@Override

public boolean updateVehicle(Vehicle vehicle) throws VehicleNotFoundException {

boolean updated = false;

try {

PreparedStatement pstmt = con.prepareStatement(

"UPDATE Vehicle SET Make=?, Model=?, Year=?, Color=?, RegistrationNumber=?, Availability=?, DailyRate=? WHERE VehicleID=?"

);

pstmt.setString(1, vehicle.getMake());

pstmt.setString(2, vehicle.getModel());

pstmt.setInt(3, vehicle.getYear());

pstmt.setString(4, vehicle.getColor());

pstmt.setString(5, vehicle.getRegistrationNumber());

pstmt.setBoolean(6, vehicle.isAvailability());

pstmt.setDouble(7, vehicle.getDailyRate());

pstmt.setInt(8, vehicle.getVehicleID());

int rows = pstmt.executeUpdate();

if (rows > 0) {

updated = true;

} else {

throw new VehicleNotFoundException("Vehicle with ID " + vehicle.getVehicleID() + " not found for update.");

}

} catch (SQLException e) {

System.out.println("Error while updating vehicle");

e.printStackTrace();

}

return updated;

}

@Override

public boolean removeVehicle(int vehicleId) throws VehicleNotFoundException {

boolean deleted = false;

try {

PreparedStatement pstmt = con.prepareStatement("DELETE FROM Vehicle WHERE VehicleID=?");

pstmt.setInt(1, vehicleId);

int rows = pstmt.executeUpdate();

if (rows > 0) {

deleted = true;

} else {

throw new VehicleNotFoundException("Vehicle with ID " + vehicleId + " not found for deletion.");

}

} catch (SQLException e) {

System.out.println("Error while deleting vehicle");

e.printStackTrace();

}

return deleted;

}

}

**IReservationService**

package dao;

import java.util.List;

import entity.Reservation;

import exception.ReservationException;

public interface IReservationService {

Reservation getReservationById(int reservationId) throws ReservationException;

List<Reservation> getReservationsByCustomerId(int customerId) throws ReservationException;

boolean createReservation(Reservation reservationData) throws ReservationException;

boolean updateReservation(Reservation reservationData) throws ReservationException;

boolean cancelReservation(int reservationId) throws ReservationException;

}

**ReservationService**

**package** dao;

**import** java.sql.\*;

**import** java.util.ArrayList;

**import** java.util.List;

**import** entity.Reservation;

**import** exception.DatabaseConnectionException;

**import** exception.ReservationException;

**import** util.DBConnUtil;

**public** **class** ReservationService **implements** IReservationService {

**private** Connection con;

**public** ReservationService() **throws** DatabaseConnectionException {

**super**();

con = DBConnUtil.*getDbConnection*();

}

@Override

**public** Reservation getReservationById(**int** reservationId) **throws** ReservationException {

Reservation reservation = **null**;

**try** {

PreparedStatement pstmt = con.prepareStatement("SELECT \* FROM Reservation WHERE ReservationID = ?");

pstmt.setInt(1, reservationId);

ResultSet rs = pstmt.executeQuery();

**if** (rs.next()) {

reservation = **new** Reservation(

rs.getInt("ReservationID"),

rs.getInt("CustomerID"),

rs.getInt("VehicleID"),

rs.getTimestamp("StartDate"),

rs.getTimestamp("EndDate"),

rs.getDouble("TotalCost"),

rs.getString("Status")

);

} **else** {

**throw** **new** ReservationException("Reservation with ID " + reservationId + " not found.");

}

} **catch** (SQLException e) {

e.printStackTrace();

**throw** **new** ReservationException("Error retrieving reservation by ID.");

}

**return** reservation;

}

@Override

**public** List<Reservation> getReservationsByCustomerId(**int** customerId) **throws** ReservationException {

List<Reservation> reservations = **new** ArrayList<>();

**try** {

PreparedStatement pstmt = con.prepareStatement("SELECT \* FROM Reservation WHERE CustomerID = ?");

pstmt.setInt(1, customerId);

ResultSet rs = pstmt.executeQuery();

**while** (rs.next()) {

Reservation reservation = **new** Reservation(

rs.getInt("ReservationID"),

rs.getInt("CustomerID"),

rs.getInt("VehicleID"),

rs.getTimestamp("StartDate"),

rs.getTimestamp("EndDate"),

rs.getDouble("TotalCost"),

rs.getString("Status")

);

reservations.add(reservation);

}

**if** (reservations.isEmpty()) {

**throw** **new** ReservationException("No reservations found for customer ID " + customerId);

}

} **catch** (SQLException e) {

e.printStackTrace();

**throw** **new** ReservationException("Error retrieving reservations for customer ID " + customerId);

}

**return** reservations;

}

@Override

**public** **boolean** createReservation(Reservation reservationData) **throws** ReservationException {

**try** {

PreparedStatement checkStmt = con.prepareStatement("SELECT \* FROM Reservation WHERE VehicleID = ? AND Status = 'Reserved' AND StartDate <= ? AND EndDate >= ?");

checkStmt.setInt(1, reservationData.getVehicleID());

checkStmt.setTimestamp(2, **new** java.sql.Timestamp(reservationData.getStartDate().getTime()));

checkStmt.setTimestamp(3, **new** java.sql.Timestamp(reservationData.getEndDate().getTime()));

ResultSet checkResult = checkStmt.executeQuery();

**if** (checkResult.next()) {

**throw** **new** ReservationException("Vehicle is already reserved for the selected period.");

}

PreparedStatement pstmt = con.prepareStatement("INSERT INTO Reservation (CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status) VALUES (?, ?, ?, ?, ?, ?)");

pstmt.setInt(1, reservationData.getCustomerID());

pstmt.setInt(2, reservationData.getVehicleID());

pstmt.setTimestamp(3, **new** java.sql.Timestamp(reservationData.getStartDate().getTime()));

pstmt.setTimestamp(4, **new** java.sql.Timestamp(reservationData.getEndDate().getTime()));

pstmt.setDouble(5, reservationData.getTotalCost());

pstmt.setString(6, reservationData.getStatus());

**int** rows = pstmt.executeUpdate();

**return** rows > 0;

} **catch** (SQLException e) {

e.printStackTrace();

**throw** **new** ReservationException("Error creating reservation.");

}

}

@Override

**public** **boolean** updateReservation(Reservation reservationData) **throws** ReservationException {

**try** {

PreparedStatement pstmt = con.prepareStatement("UPDATE Reservation SET StartDate = ?, EndDate = ?, TotalCost = ?, Status = ? WHERE ReservationID = ?");

pstmt.setTimestamp(1, **new** java.sql.Timestamp(reservationData.getStartDate().getTime())); // Convert StartDate to Timestamp

pstmt.setTimestamp(2, **new** java.sql.Timestamp(reservationData.getEndDate().getTime()));

pstmt.setDouble(3, reservationData.getTotalCost());

pstmt.setString(4, reservationData.getStatus());

pstmt.setInt(5, reservationData.getReservationID());

**int** rows = pstmt.executeUpdate();

**return** rows > 0;

} **catch** (SQLException e) {

e.printStackTrace();

**throw** **new** ReservationException("Error updating reservation.");

}

}

@Override

**public** **boolean** cancelReservation(**int** reservationId) **throws** ReservationException {

**try** {

PreparedStatement pstmt = con.prepareStatement("UPDATE Reservation SET Status = 'Cancelled' WHERE ReservationID = ?");

pstmt.setInt(1, reservationId);

**int** rows = pstmt.executeUpdate();

**return** rows > 0;

} **catch** (SQLException e) {

e.printStackTrace();

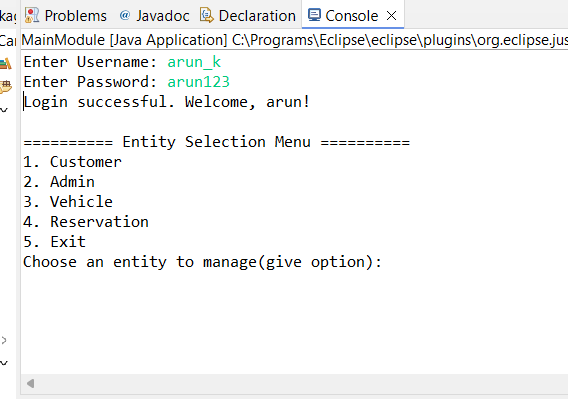
**throw** **new** ReservationException("Error cancelling reservation.");

}

}

}

**UserAuthentication**



1. **Test customer authentication with invalid credentials**

package dao.test;

import static org.junit.Assert.\*;

import org.junit.Test;

import exception.DatabaseConnectionException;

import exception.UserNotFoundException;

import dao.AuthenticationService;

public class AuthenticationServiceTest {

@Test

public void test() {

try {

AuthenticationService authService = new AuthenticationService();

authService.authenticateUser("invalidUser", "invalidPass");

fail("Expected UserNotFoundException for invalid credentials.");

} catch (UserNotFoundException e) {

assertTrue(e.getMessage().contains("Invalid username or password") ||

e.getMessage().contains("Error during authentication"));

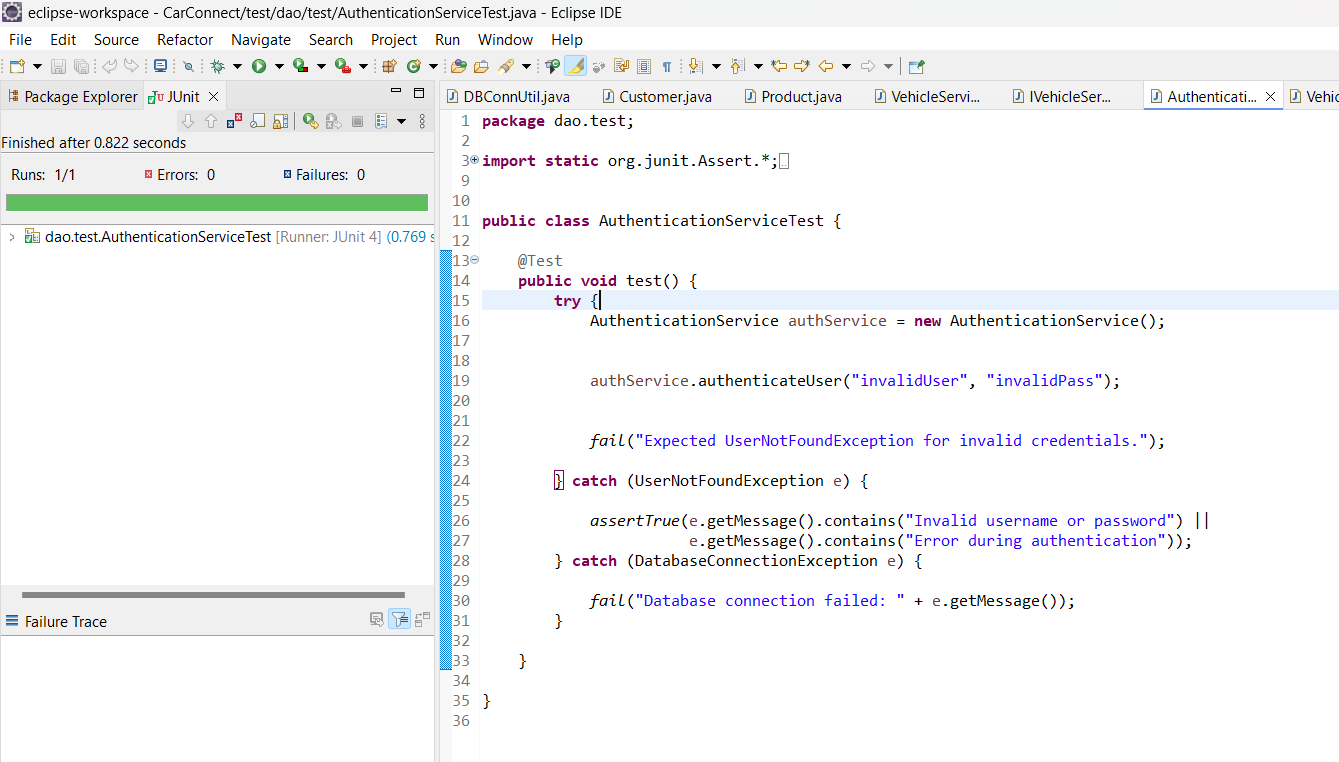
} catch (DatabaseConnectionException e) {

fail("Database connection failed: " + e.getMessage());

}

}

}



1. **Test updating customer information.**

**package** dao.test;

**import** **static** org.junit.Assert.\*;

**import** org.junit.Before;

**import** org.junit.Test;

**import** dao.CustomerService;

**import** entity.Customer;

**import** exception.DatabaseConnectionException;

**import** exception.InvalidInputException;

**import** java.util.Date;

**public** **class** CustomerUpdateTest {

**private** CustomerService customerService;

@Before

**public** **void** setUp() **throws** DatabaseConnectionException {

customerService = **new** CustomerService();

}

@Test

**public** **void** testUpdateCustomer() {

**try** {

**int** existingCustomerId = 1;

Customer updatedCustomer = **new** Customer();

updatedCustomer.setCustomerID(existingCustomerId);

updatedCustomer.setFirstName("Johnny");

updatedCustomer.setLastName("Dough");

updatedCustomer.setEmail("johnny.dough@example.com");

updatedCustomer.setPhoneNumber("0987654321");

updatedCustomer.setAddress("5678 Oak St");

updatedCustomer.setUsername("johnny123");

updatedCustomer.setPassword("newpassword123");

updatedCustomer.setRegistrationDate(**new** Date());

**boolean** result = customerService.updateCustomer(updatedCustomer);

*assertTrue*("Customer update failed", result);

Customer retrieved = customerService.getCustomerById(existingCustomerId);

*assertEquals*("Johnny", retrieved.getFirstName());

*assertEquals*("Dough", retrieved.getLastName());

*assertEquals*("johnny.dough@example.com", retrieved.getEmail());

*assertEquals*("0987654321", retrieved.getPhoneNumber());

*assertEquals*("5678 Oak St", retrieved.getAddress());

*assertEquals*("johnny123", retrieved.getUsername());

*assertEquals*("newpassword123", retrieved.getPassword());

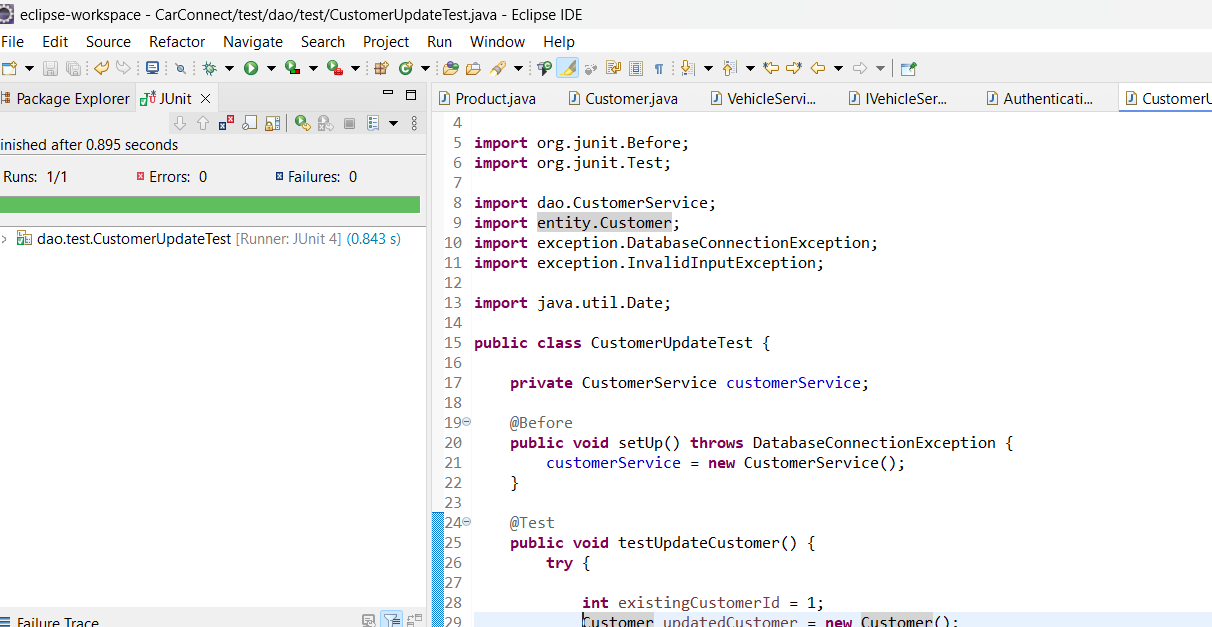
} **catch** (InvalidInputException | DatabaseConnectionException e) {

*fail*("Exception during test: " + e.getMessage());

}

}

}



1. **Test adding a new vehicle.**

package dao.test;

import static org.junit.Assert.\*;

import org.junit.Test;

import dao.VehicleService;

import entity.Vehicle;

import exception.InvalidInputException;

import exception.DatabaseConnectionException;

public class VehicleAddTest {

@Test

public void test() {

try {

VehicleService vehicleService = new VehicleService();

Vehicle vehicle = new Vehicle(0, "Swift", "Maruti", 2021, "White", "TN01AB1234", true, 1200.0);

boolean result = vehicleService.addVehicle(vehicle);

assertTrue("Vehicle should be added successfully", result);

} catch (InvalidInputException e) {

fail("Validation failed: " + e.getMessage());

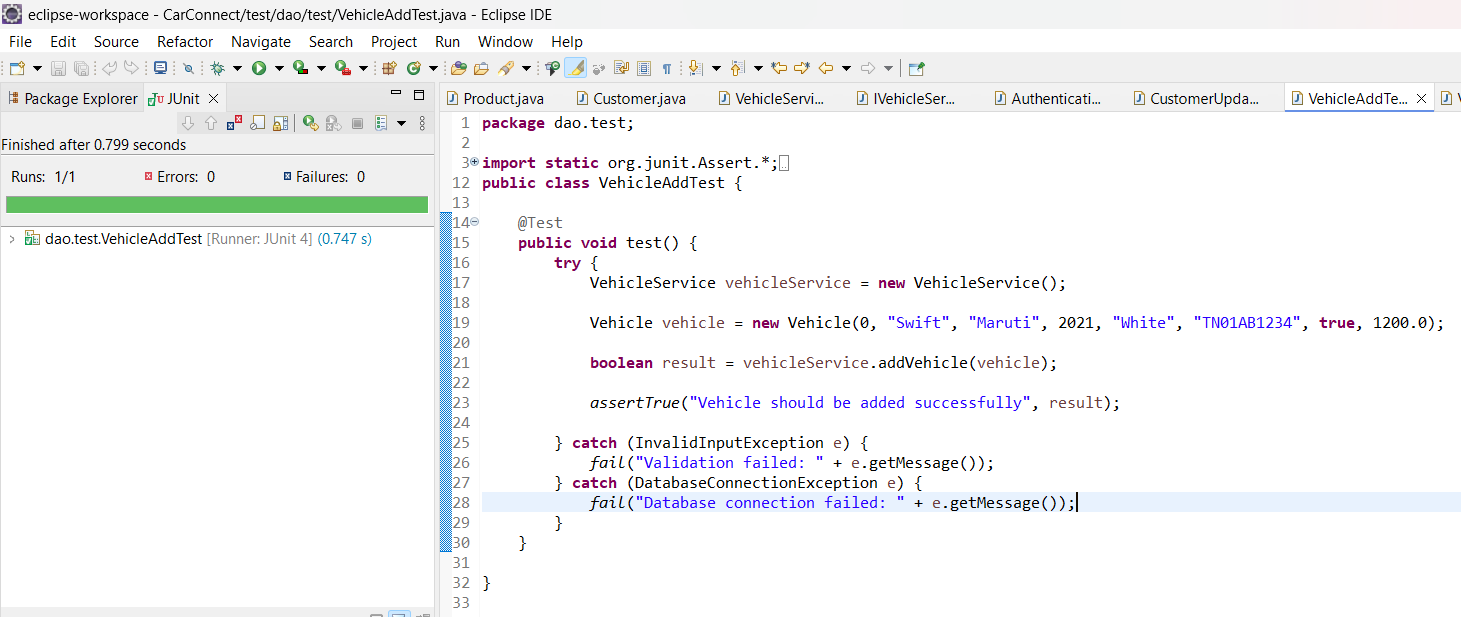
} catch (DatabaseConnectionException e) {

fail("Database connection failed: " + e.getMessage());

}

}

}



1. **Test updating vehicle details.**

**package** dao.test;

**import** org.junit.Test;

**import** dao.VehicleService;

**import** entity.Vehicle;

**import** exception.VehicleNotFoundException;

**import** exception.DatabaseConnectionException;

**import** **static** org.junit.Assert.\*;

**public** **class** VehicleUpdateTest {

@Test

**public** **void** test() {

**try**{

VehicleService vehicleService = **new** VehicleService();

Vehicle vehicle = **new** Vehicle(1, "Alto", "Maruti", 2023, "Red", "TN10AB4321", **true**, 1100.0);

**boolean** result = vehicleService.updateVehicle(vehicle);

*assertTrue*("Vehicle should be updated successfully", result);

} **catch** (VehicleNotFoundException e) {

*fail*("Vehicle not found for update: " + e.getMessage());

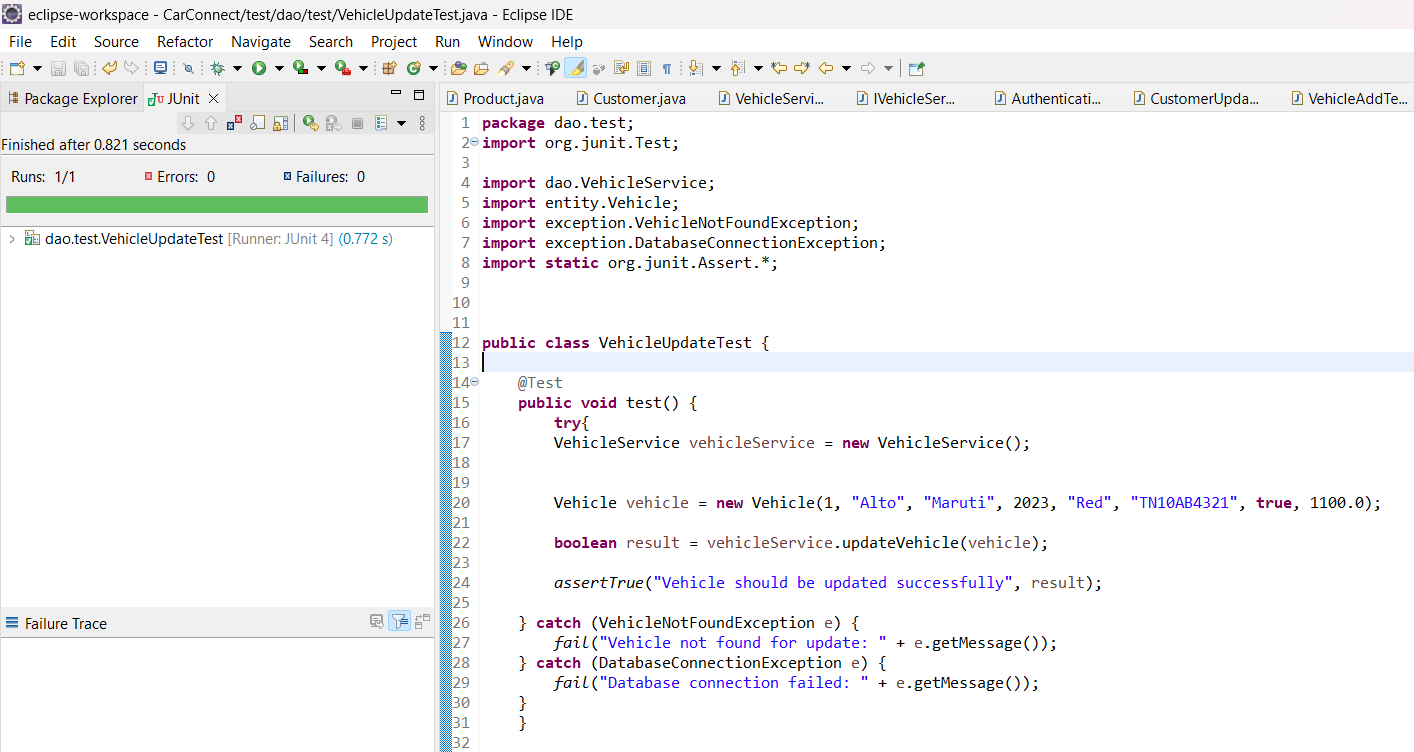
} **catch** (DatabaseConnectionException e) {

*fail*("Database connection failed: " + e.getMessage());

}

}

}



1. **Test getting a list of available vehicles.**

**package** dao.test;

**import** **static** org.junit.Assert.\*;

**import** java.util.HashMap;

**import** org.junit.Before;

**import** org.junit.Test;

**import** dao.VehicleService;

**import** entity.Vehicle;

**import** exception.DatabaseConnectionException;

**public** **class** AvailableVehiclesTest {

**private** VehicleService vehicleService;

@Before

**public** **void** setUp() **throws** DatabaseConnectionException {

vehicleService = **new** VehicleService();

}

@Test

**public** **void** testGetAvailableVehicles() {

HashMap<Integer, Vehicle> availableVehicles = vehicleService.getAvailableVehicles();

*assertNotNull*("Returned map should not be null", availableVehicles);

**for** (Vehicle v : availableVehicles.values()) {

*assertTrue*("Vehicle should be marked available", v.isAvailability());

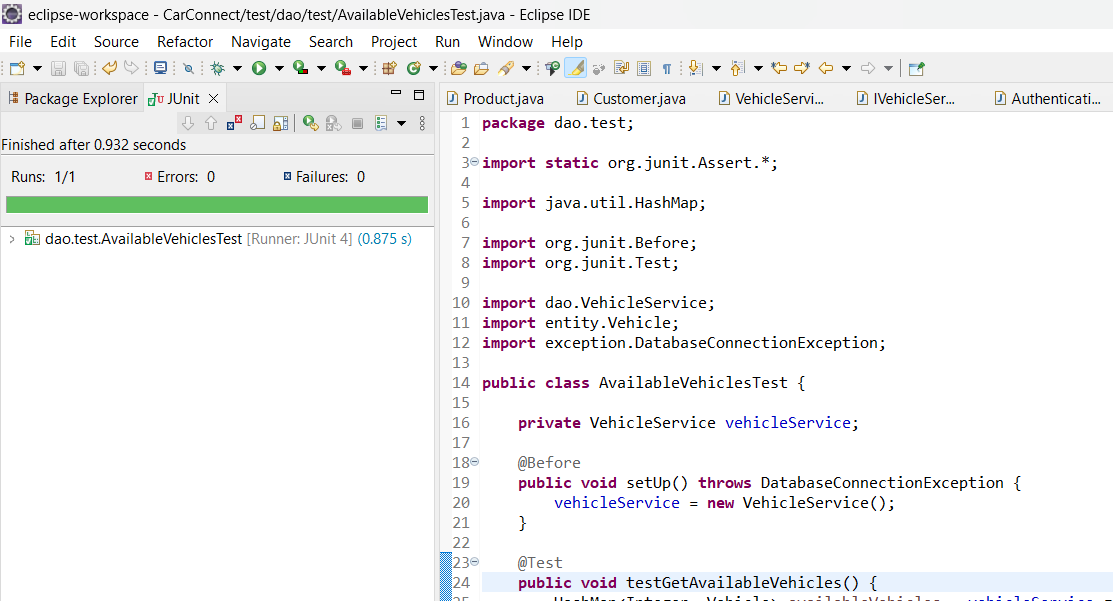
*assertNotNull*("Model should not be null", v.getModel());

*assertTrue*("VehicleID should be positive", v.getVehicleID() > 0);

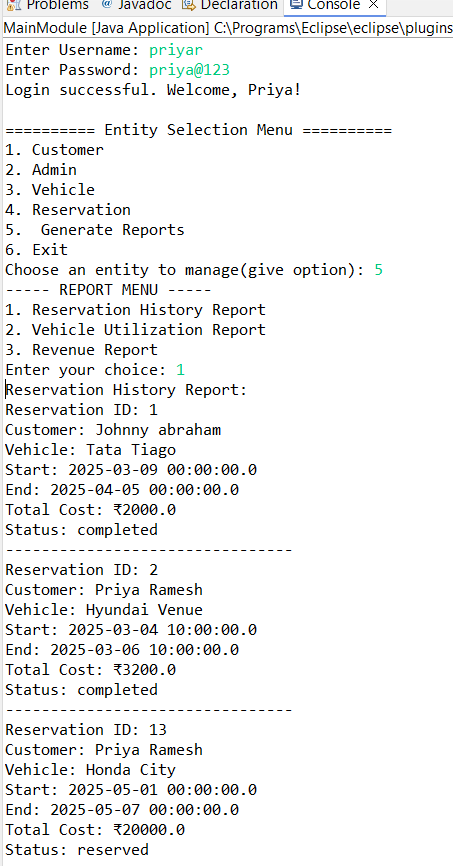
}

}

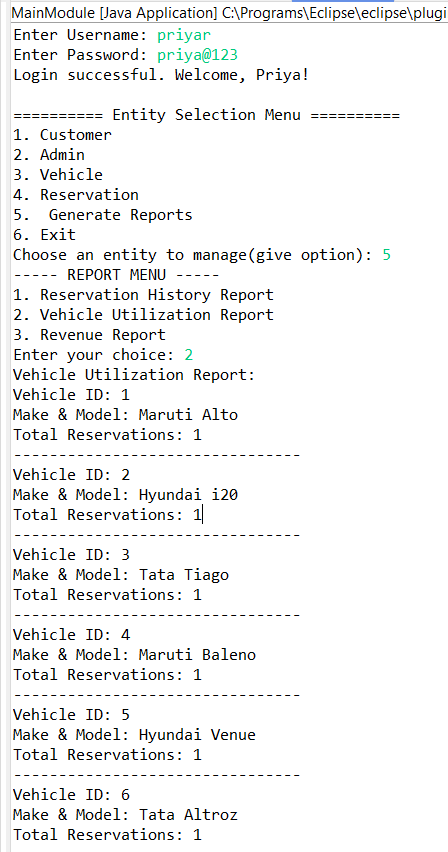
}



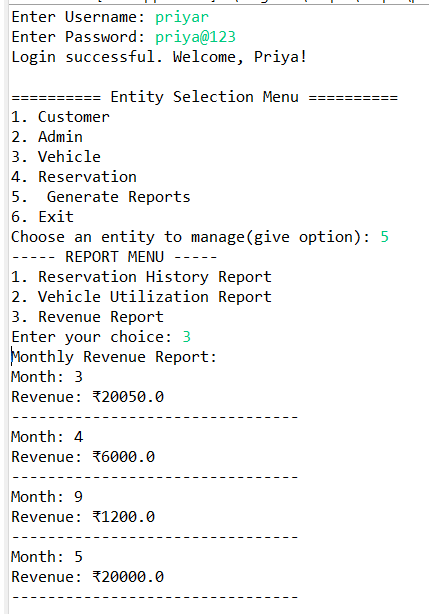
**Reservation History report**



**Vehicle Utilization Report**



**Monthly Revenue Report:**



**ReportGenerator class**

**package** dao;

**import** java.sql.\*;

**import** util.DBConnUtil;

**import** exception.DatabaseConnectionException;

**public** **class** ReportGenerator {

**private** Connection con;

**public** ReportGenerator() **throws** DatabaseConnectionException {

con = DBConnUtil.*getDbConnection*();

}

**public** **void** generateReservationHistory() {

String query = "SELECT r.ReservationID, c.FirstName, c.LastName, v.Model, v.Make, r.StartDate, r.EndDate, r.TotalCost, r.Status " +

"FROM Reservation r " +

"JOIN Customer c ON r.CustomerID = c.CustomerID " +

"JOIN Vehicle v ON r.VehicleID = v.VehicleID";

**try** (Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery(query)) {

System.***out***.println("Reservation History Report:");

**while** (rs.next()) {

System.***out***.println("Reservation ID: " + rs.getInt("ReservationID"));

System.***out***.println("Customer: " + rs.getString("FirstName") + " " + rs.getString("LastName"));

System.***out***.println("Vehicle: " + rs.getString("Make") + " " + rs.getString("Model"));

System.***out***.println("Start: " + rs.getTimestamp("StartDate"));

System.***out***.println("End: " + rs.getTimestamp("EndDate"));

System.***out***.println("Total Cost: ₹" + rs.getDouble("TotalCost"));

System.***out***.println("Status: " + rs.getString("Status"));

System.***out***.println("--------------------------------");

}

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**public** **void** generateVehicleUtilization() {

String query = "SELECT v.VehicleID, v.Make, v.Model, COUNT(r.ReservationID) AS TotalReservations " +

"FROM Vehicle v " +

"LEFT JOIN Reservation r ON v.VehicleID = r.VehicleID " +

"GROUP BY v.VehicleID";

**try** (Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery(query)) {

System.***out***.println("Vehicle Utilization Report:");

**while** (rs.next()) {

System.***out***.println("Vehicle ID: " + rs.getInt("VehicleID"));

System.***out***.println("Make & Model: " + rs.getString("Make") + " " + rs.getString("Model"));

System.***out***.println("Total Reservations: " + rs.getInt("TotalReservations"));

System.***out***.println("--------------------------------");

}

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**public** **void** generateRevenueReport() {

String query = "SELECT MONTH(StartDate) AS Month, SUM(TotalCost) AS MonthlyRevenue " +

"FROM Reservation " +

"GROUP BY MONTH(StartDate)";

**try** (Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery(query)) {

System.***out***.println("Monthly Revenue Report:");

**while** (rs.next()) {

System.***out***.println("Month: " + rs.getInt("Month"));

System.***out***.println("Revenue: ₹" + rs.getDouble("MonthlyRevenue"));

System.***out***.println("--------------------------------");

}

} **catch** (SQLException e) {

e.printStackTrace();

}

}

}

**SQL**

CREATE DATABASE CarConnectDB;

USE CarConnectDB;

CREATE TABLE Customer (

CustomerID INT PRIMARY KEY AUTO\_INCREMENT,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Email VARCHAR(100),

PhoneNumber VARCHAR(20),

Address VARCHAR(255),

Username VARCHAR(50) UNIQUE,

Password VARCHAR(100),

RegistrationDate DATE

);

desc Customer;

CREATE TABLE Vehicle (

VehicleID INT PRIMARY KEY AUTO\_INCREMENT,

Model VARCHAR(100),

Make VARCHAR(100),

Year INT,

Color VARCHAR(30),

RegistrationNumber VARCHAR(50) UNIQUE,

Availability BOOLEAN,

DailyRate DECIMAL(10,2)

);

desc Vehicle;

CREATE TABLE Reservation (

ReservationID INT PRIMARY KEY AUTO\_INCREMENT,

CustomerID INT,

VehicleID INT,

StartDate DATETIME,

EndDate DATETIME,

TotalCost DECIMAL(10,2),

Status VARCHAR(20),

FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),

FOREIGN KEY (VehicleID) REFERENCES Vehicle(VehicleID)

);

desc Reservation;

CREATE TABLE Admin (

AdminID INT PRIMARY KEY AUTO\_INCREMENT,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Email VARCHAR(100),

PhoneNumber VARCHAR(20),

Username VARCHAR(50) UNIQUE,

Password VARCHAR(100),

Role VARCHAR(50),

JoinDate DATE

);

desc Admin;

INSERT INTO Customer (FirstName, LastName, Email, PhoneNumber, Address, Username, Password, RegistrationDate)

VALUES

('Arun', 'Kumar', 'arun.kumar@gmail.com', '9840012345', 'Chennai, Tamil Nadu', 'arun\_k', 'arun123', '2024-12-01'),

('Priya', 'Ramesh', 'priya.ramesh@yahoo.com', '9876543210', 'Coimbatore, Tamil Nadu', 'priyar', 'priya@123', '2024-12-02'),

('Karthik', 'Murali', 'karthik.m@rediffmail.com', '9898989898', 'Madurai, Tamil Nadu', 'karthikm', 'kmPass!21', '2024-12-03'),

('Divya', 'Shree', 'divya.shree@gmail.com', '9797979797', 'Tiruchirappalli, Tamil Nadu', 'divyashree', 'dsSecure44', '2024-12-04'),

('Sundar', 'Raj', 'sundar.raj@hotmail.com', '9888877665', 'Salem, Tamil Nadu', 'sundarraj', 'sundar@321', '2024-12-05'),

('Meena', 'Kumari', 'meena.k@gmail.com', '9787865432', 'Erode, Tamil Nadu', 'meenak', 'meena#pass', '2024-12-06'),

('Vignesh', 'Ram', 'vignesh.r@yahoo.com', '9767676767', 'Vellore, Tamil Nadu', 'vignesh\_r', 'vigRam007', '2024-12-07'),

('Lakshmi', 'Devi', 'lakshmi.d@gmail.com', '9876567890', 'Thoothukudi, Tamil Nadu', 'lakshmid', 'lakshmiD$', '2024-12-08'),

('Naveen', 'Sankar', 'naveen.s@gmail.com', '9654321098', 'Tirunelveli, Tamil Nadu', 'naveens', 'naveen@998', '2024-12-09'),

('Anitha', 'Prabha', 'anitha.p@yahoo.com', '9567890123', 'Kanchipuram, Tamil Nadu', 'anithap', 'anitha##1122', '2024-12-10');

select \* from Customer;

INSERT INTO Vehicle (Model, Make, Year, Color, RegistrationNumber, Availability, DailyRate)

VALUES

('Swift', 'Maruti', 2020, 'White', 'TN01AB1234', TRUE, 1200.00),

('i20', 'Hyundai', 2021, 'Red', 'TN02CD5678', FALSE, 1500.00),

('Tiago', 'Tata', 2019, 'Blue', 'TN03EF9012', TRUE, 1100.00),

('Baleno', 'Maruti', 2022, 'Grey', 'TN04GH3456', FALSE, 1400.00),

('Venue', 'Hyundai', 2021, 'Black', 'TN05IJ7890', TRUE, 1600.00),

('Altroz', 'Tata', 2020, 'Silver', 'TN06KL1234', FALSE, 1150.00),

('Seltos', 'Kia', 2022, 'White', 'TN07MN5678', TRUE, 1800.00),

('City', 'Honda', 2020, 'Red', 'TN08OP9012', TRUE, 1700.00),

('Creta', 'Hyundai', 2021, 'Blue', 'TN09QR3456', FALSE, 1750.00),

('Dzire', 'Maruti', 2019, 'Black', 'TN10ST7890', TRUE, 1250.00);

select \* from Vehicle ;

INSERT INTO Reservation (CustomerID, VehicleID, StartDate, EndDate, TotalCost, Status)

VALUES

(1, 3, '2025-03-01 09:00:00', '2025-03-03 09:00:00', 2200.00, 'completed'),

(2, 5, '2025-03-04 10:00:00', '2025-03-06 10:00:00', 3200.00, 'completed'),

(3, 1, '2025-03-10 08:30:00', '2025-03-12 08:30:00', 2400.00, 'confirmed'),

(4, 2, '2025-03-15 09:00:00', '2025-03-16 09:00:00', 1500.00, 'pending'),

(5, 7, '2025-03-18 11:00:00', '2025-03-20 11:00:00', 3600.00, 'completed'),

(6, 4, '2025-03-21 09:00:00', '2025-03-23 09:00:00', 2800.00, 'confirmed'),

(7, 6, '2025-03-25 10:00:00', '2025-03-26 10:00:00', 1150.00, 'completed'),

(8, 8, '2025-03-28 08:00:00', '2025-03-30 08:00:00', 3400.00, 'cancelled'),

(9, 9, '2025-04-01 07:30:00', '2025-04-03 07:30:00', 3500.00, 'confirmed'),

(10, 10, '2025-04-04 09:00:00', '2025-04-06 09:00:00', 2500.00, 'pending');

select \* from Reservation ;

INSERT INTO Admin (FirstName, LastName, Email, PhoneNumber, Username, Password, Role, JoinDate)

VALUES

('Ravi', 'Subramanian', 'ravi.s@carconnect.in', '9840011122', 'ravi\_admin', 'admin@123', 'super admin', '2022-01-10'),

('Meena', 'Iyer', 'meena.i@carconnect.in', '9840022233', 'meena\_fleet', 'fleet@456', 'fleet manager', '2022-02-15'),

('Arun', 'Raj', 'arun.raj@carconnect.in', '9840033344', 'arun\_mgr', 'manager@789', 'fleet manager', '2022-03-20'),

('Divya', 'Bharathi', 'divya.b@carconnect.in', '9840044455', 'divya\_admin', 'pass@divya', 'super admin', '2022-04-05'),

('Sathish', 'Kumar', 'sathish.k@carconnect.in', '9840055566', 'sathish\_ops', 'sathish@ops', 'operations', '2022-05-10'),

('Priya', 'Sundar', 'priya.s@carconnect.in', '9840066677', 'priya\_fleet', 'priya@123', 'fleet manager', '2022-06-12'),

('Karthik', 'Venkat', 'karthik.v@carconnect.in', '9840077788', 'karthik\_admin', 'karthik@admin', 'super admin', '2022-07-18'),

('Lakshmi', 'Devi', 'lakshmi.d@carconnect.in', '9840088899', 'lakshmi\_mgr', 'lakshmi@456', 'fleet manager', '2022-08-20'),

('Vijay', 'Kumar', 'vijay.k@carconnect.in', '9840099900', 'vijay\_ops', 'vijay@ops', 'operations', '2022-09-25'),

('Anitha', 'Ravi', 'anitha.r@carconnect.in', '9840101010', 'anitha\_admin', 'anitha@321', 'super admin', '2022-10-30');

select \* from Admin;