NAME: THILAK RAAJ R A

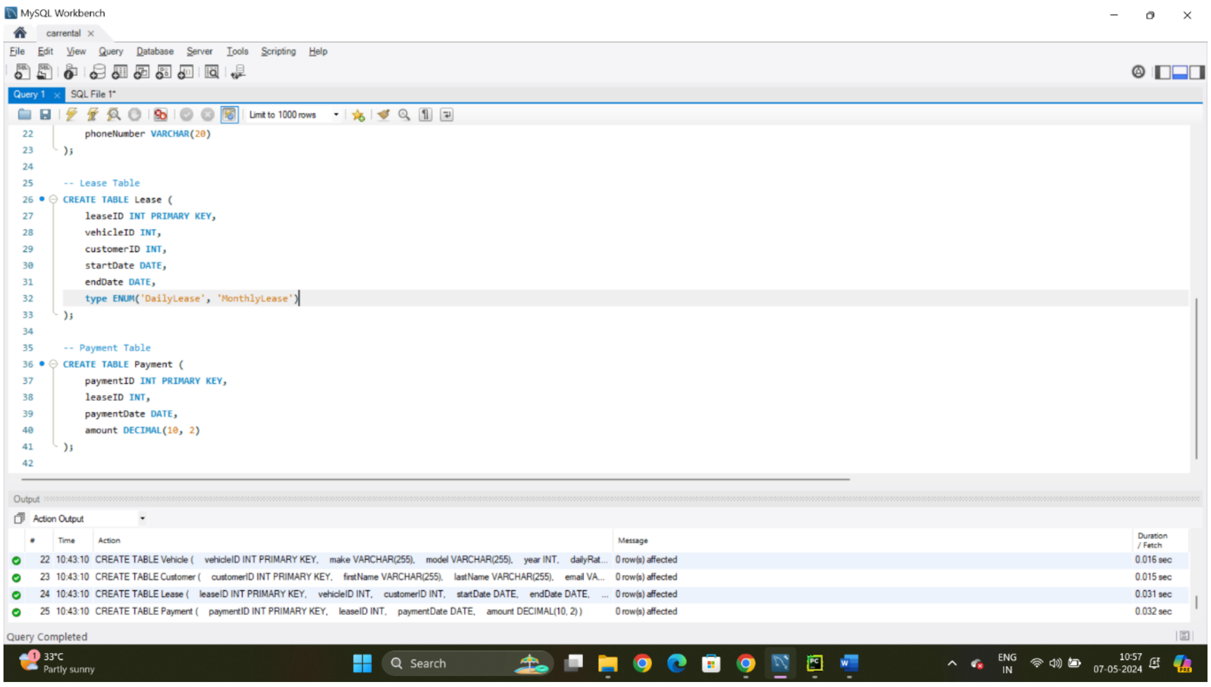
TOPIC: CAR RENTAL CASESTUDY

1)Create following tables in SQL Schema with appropriate class and write the unit test case for the Car Rental application.

Schema Design: 1. Vehicle Table: • vehicleID (Primary Key) • make • model • year • dailyRate • status (available, notAvailable) • passengerCapacity • engineCapacity 2. Customer Table: • customerID (Primary Key) • firstName • lastName • email • phoneNumber 3. Lease Table: • leaseID (Primary Key) • vehicleID (Foreign Key referencing Vehicle Table) • customerID (Foreign Key referencing Customer Table) • startDate • endDate • type (to distinguish between DailyLease and MonthlyLease) 4. Payment Table: • paymentID (Primary Key) • leaseID (Foreign Key referencing Lease Table) • paymentDate • amount

CODE:

```
create database carrental;

use carrental;


-- Vehicle Table

CREATE TABLE Vehicle (

   vehicleID INT PRIMARY KEY,

   make VARCHAR(255),

   model VARCHAR(255),

   year INT,

   dailyRate DECIMAL(10, 2),

   status ENUM('available', 'notAvailable'),

   passengerCapacity INT,

   engineCapacity VARCHAR(50)

);


-- Customer Table

CREATE TABLE Customer (

   customerID INT PRIMARY KEY,

   firstName VARCHAR(255),

   lastName VARCHAR(255),
    email VARCHAR(255),
    phoneNumber VARCHAR(20)
);

-- Lease Table
CREATE TABLE Lease (
    leaseID INT PRIMARY KEY,
    vehicleID INT,
    customerID INT,
    startDate DATE,
    endDate DATE,
    type ENUM('DailyLease', 'MonthlyLease')
);

-- Payment Table
CREATE TABLE Payment (
    paymentID INT PRIMARY KEY,
    leaseID INT,
    paymentDate DATE,
    amount DECIMAL(10, 2)
);
```



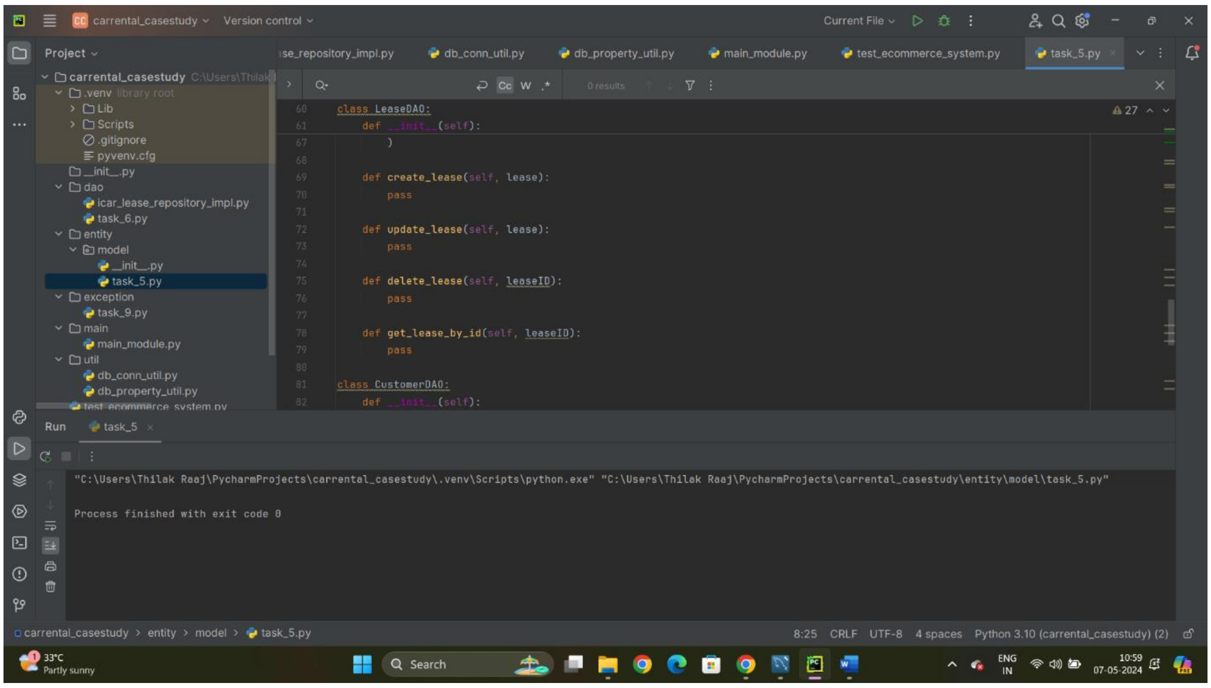
```sql
    phoneNumber VARCHAR(20)
);

-- Lease Table
CREATE TABLE Lease (
    leaseID INT PRIMARY KEY,
    vehicleID INT,
    customerID INT,
    startDate DATE,
    endDate DATE,
    type ENUM('DailyLease', 'MonthlyLease')
);

-- Payment Table
CREATE TABLE Payment (
    paymentID INT PRIMARY KEY,
    leaseID INT,
    paymentDate DATE,
    amount DECIMAL(10, 2)
);
```

```
#   Time      Action                                                                                                   Message           Duration / Fetch
22  10:43:10  CREATE TABLE Vehicle (   vehicleID INT PRIMARY KEY,   make VARCHAR(255),   model VARCHAR(255),   year INT,   dailyRat...   0 row(s) affected   0.016 sec
23  10:43:10  CREATE TABLE Customer (   customerID INT PRIMARY KEY,   firstName VARCHAR(255),   lastName VARCHAR(255),   email VA...   0 row(s) affected   0.015 sec
24  10:43:10  CREATE TABLE Lease (   leaseID INT PRIMARY KEY,   vehicleID INT,   customerID INT,   startDate DATE,   endDate DATE,   ...   0 row(s) affected   0.031 sec
25  10:43:10  CREATE TABLE Payment (   paymentID INT PRIMARY KEY,   leaseID INT,   paymentDate DATE,   amount DECIMAL(10, 2) )   0 row(s) affected   0.032 sec
Query Completed
```

5. Create the model/entity classes corresponding to the schema within package entity with variables declared private, constructors(default and parametrized) and getters,setters)

```python
import mysql.connector

class Vehicle:
    def __init__(self, id, make, model, year, daily_rate, status,
passenger_capacity, engine_capacity):
        self.id = id
        self.make = make
        self.model = model
        self.year = year
        self.daily_rate = daily_rate
        self.status = status
        self.passenger_capacity = passenger_capacity
        self.engine_capacity = engine_capacity

class Customer:
    def __init__(self, customerID, firstName, lastName, email,
phoneNumber):
        self.customerID = customerID
        self.firstName = firstName
        self.lastName = lastName
        self.email = email
        self.phoneNumber = phoneNumber

class Lease:
    def __init__(self, leaseID, vehicleID, customerID, startDate, endDate,
type):
        self.leaseID = leaseID
        self.vehicleID = vehicleID
        self.customerID = customerID
        self.startDate = startDate
        self.endDate = endDate
        self.type = type

class Payment:
    def __init__(self, paymentID, leaseID, paymentDate, amount):
        self.paymentID = paymentID
        self.leaseID = leaseID
        self.paymentDate = paymentDate
        self.amount = amount

class CarDAO:
    def __init__(self):
        # Initialize database connection for car DAO
        self.connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="car_rental"
        )

    def create_car(self, vehicle):
        pass

    def update_car(self, vehicle):
        pass

    def delete_car(self, vehicleID):
        pass

    def get_car_by_id(self, vehicleID):
        pass

class LeaseDAO:
    def __init__(self):
        self.connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="car_rental"
        )

    def create_lease(self, lease):
        pass

    def update_lease(self, lease):
        pass

    def delete_lease(self, leaseID):
        pass

    def get_lease_by_id(self, leaseID):
        pass

class CustomerDAO:
    def __init__(self):
        self.connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="car_rental"
        )
```

```python
    def create_customer(self, customer):
        pass

    def update_customer(self, customer):
        pass

    def delete_customer(self, customerID):
        pass

    def get_customer_by_id(self, customerID):
        pass
```



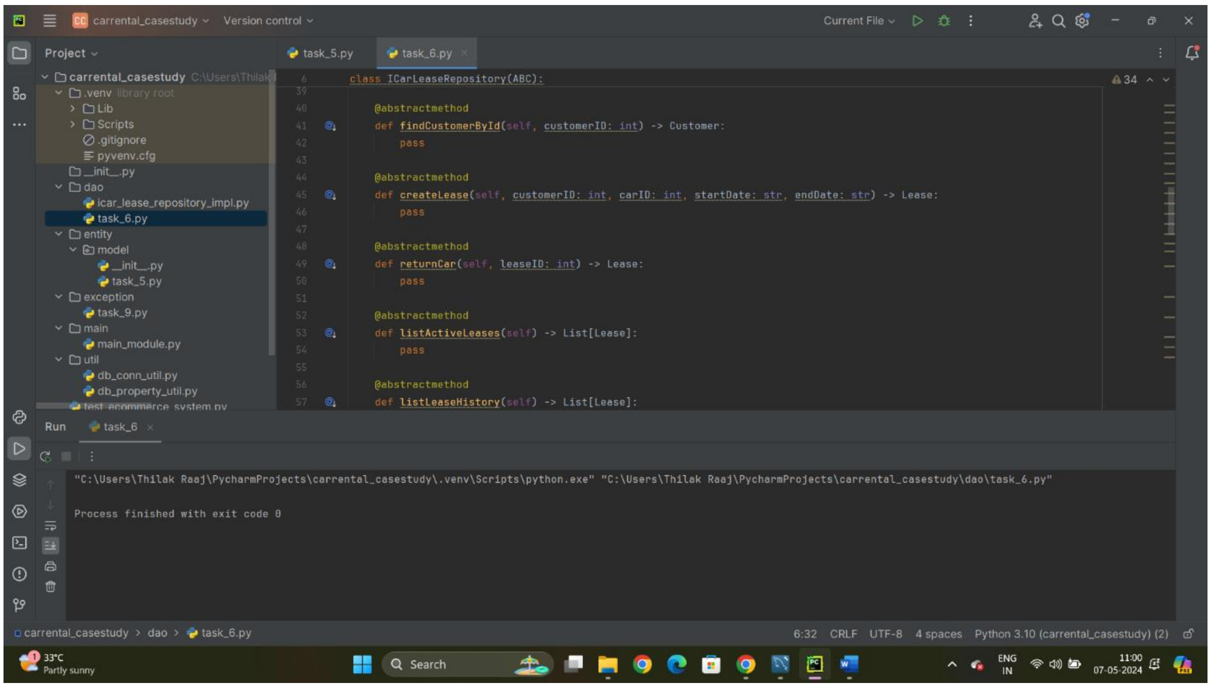
```python
class LeaseDAO:
    def __init__(self):
        )

    def create_lease(self, lease):
        pass

    def update_lease(self, lease):
        pass

    def delete_lease(self, leaseID):
        pass

    def get_lease_by_id(self, leaseID):
        pass

class CustomerDAO:
    def __init__(self):
```

```
"C:\Users\Thilak Raaj\PycharmProjects\carrental_casestudy\.venv\Scripts\python.exe" "C:\Users\Thilak Raaj\PycharmProjects\carrental_casestudy\entity\model\task_5.py"

Process finished with exit code 0
```

6. Service Provider Interface/Abstract class: Keep the interfaces and implementation classes in package dao • Create Interface for ICarLeaseRepository and add following methods which interact with database. • Car Management 1. addCar(Car car) parameter : Car return type : void 2. removeCar() parameter : carID return type : void 3. listAvailableCars() - parameter: NIL return type: return List of Car 4. listRentedCars() – return List of Car parameter: NIL return type: return List of Car 5. findCarById(int carID) – return Car if found or throw exception parameter: NIL return type: return List of Car • Customer Management 1. addCustomer(Customer customer) parameter : Customer return type : void 2. void removeCustomer(int customerID) parameter : CustomerID return type : void 3. listCustomers() parameter : NIL return type : list of customer 4. findCustomerById(int customerID) parameter : CustomerID return type : Customer • Lease Management 1. createLease() parameter : int customerID, int carID, Date startDate, Date endDate return type : Lease 2. void returnCar(); parameter : int leaseID return type : Lease info 3. List listActiveLeases(); parameter : NIL return type : Lease list 4. listLeaseHistory(); parameter : NIL return type : Lease list • Payment Handling 1. void recordPayment(); parameter : Lease lease, double amount return type : void

```python
# src/dao/task_6.py
from abc import ABC, abstractmethod
from typing import List
from entity.model.task_5 import Vehicle, Customer, Lease

class ICarLeaseRepository(ABC):

    @abstractmethod
    def addCar(self, car: Vehicle) -> None:
        pass

    @abstractmethod
    def removeCar(self, carID: int) -> None:
        pass

    @abstractmethod
    def listAvailableCars(self) -> List[Vehicle]:
        pass

    @abstractmethod
    def listRentedCars(self) -> List[Vehicle]:
        pass

    @abstractmethod
    def findCarById(self, carID: int) -> Vehicle:
        pass

    @abstractmethod
    def addCustomer(self, customer: Customer) -> None:
        pass

    @abstractmethod
    def removeCustomer(self, customerID: int) -> None:
        pass

    @abstractmethod
    def listCustomers(self) -> List[Customer]:
        pass

    @abstractmethod
    def findCustomerById(self, customerID: int) -> Customer:
        pass

    @abstractmethod
    def createLease(self, customerID: int, carID: int, startDate: str,
endDate: str) -> Lease:
        pass

    @abstractmethod
    def returnCar(self, leaseID: int) -> Lease:
        pass

    @abstractmethod
    def listActiveLeases(self) -> List[Lease]:
        pass

    @abstractmethod
    def listLeaseHistory(self) -> List[Lease]:
        pass

    @abstractmethod
    def recordPayment(self, lease: Lease, amount: float) -> None:
        pass
```

```python
class CarLeaseRepositoryImpl(ICarLeaseRepository):

    def addCar(self, car: Vehicle) -> None:
        pass

    def removeCar(self, carID: int) -> None:
        pass

    def listAvailableCars(self) -> List[Vehicle]:
        pass

    def listRentedCars(self) -> List[Vehicle]:
        pass

    def findCarById(self, carID: int) -> Vehicle:
        pass

    def addCustomer(self, customer: Customer) -> None:
        pass

    def removeCustomer(self, customerID: int) -> None:
        pass

    def listCustomers(self) -> List[Customer]:
        pass

    def findCustomerById(self, customerID: int) -> Customer:
        pass

    def createLease(self, customerID: int, carID: int, startDate: str,
endDate: str) -> Lease:
        pass

    def returnCar(self, leaseID: int) -> Lease:
        pass

    def listActiveLeases(self) -> List[Lease]:
        pass

    def listLeaseHistory(self) -> List[Lease]:
        pass

    def recordPayment(self, lease: Lease, amount: float) -> None:
        pass
```



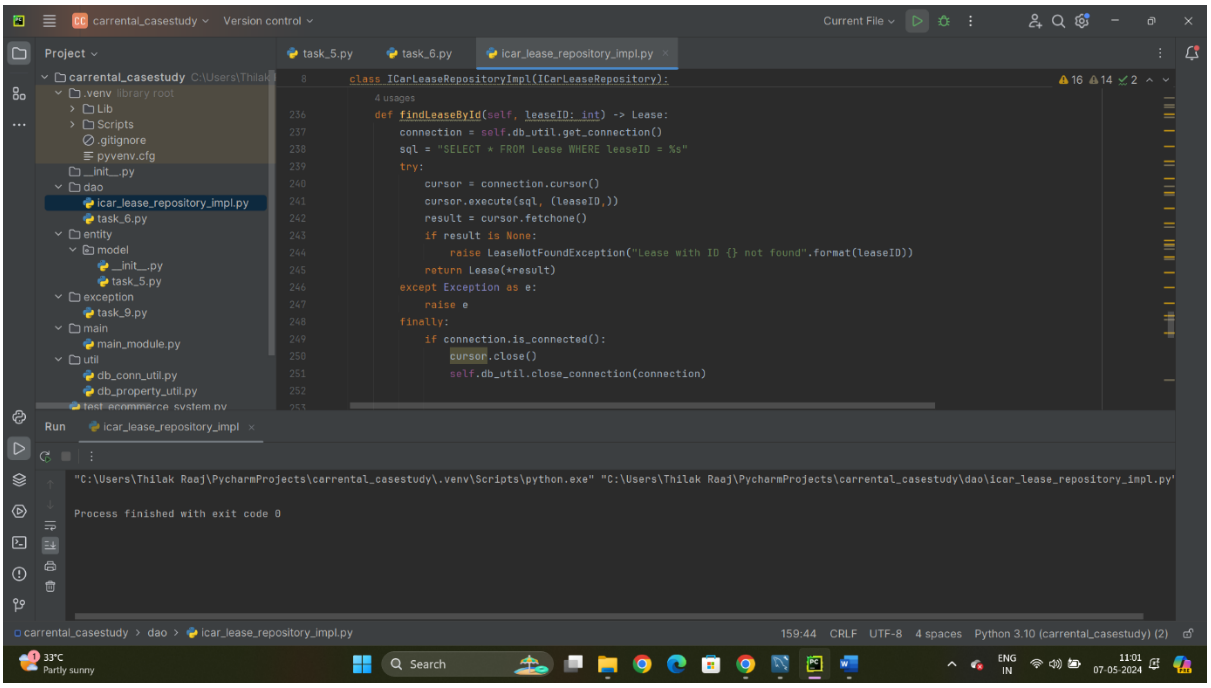
```
class ICarLeaseRepository(ABC):

    @abstractmethod
    def findCustomerById(self, customerID: int) -> Customer:
        pass

    @abstractmethod
    def createLease(self, customerID: int, carID: int, startDate: str, endDate: str) -> Lease:
        pass

    @abstractmethod
    def returnCar(self, leaseID: int) -> Lease:
        pass

    @abstractmethod
    def listActiveLeases(self) -> List[Lease]:
        pass

    @abstractmethod
    def listLeaseHistory(self) -> List[Lease]:
```

```
"C:\Users\Thilak Raaj\PycharmProjects\carrental_casestudy\.venv\Scripts\python.exe" "C:\Users\Thilak Raaj\PycharmProjects\carrental_casestudy\dao\task_6.py"

Process finished with exit code 0
```

7. Implement the above interface in a class called ICarLeaseRepositoryImpl in package dao.

```
# src/dao/icar_lease_repository_impl.py
from typing import List
from entity.model.task_5 import Vehicle, Customer, Lease
from dao.task_6 import ICarLeaseRepository
from util.db_conn_util import DBConnUtil  # Ensure this utility is
correctly defined to handle database connections
from exception.task_9 import CarNotFoundException, LeaseNotFoundException,
CustomerNotFoundException

class ICarLeaseRepositoryImpl(ICarLeaseRepository):
    def __init__(self):
        self.db_util = DBConnUtil()

    def addCar(self, vehicle):
        connection = self.db_util.get_connection()
        sql = "INSERT INTO Vehicle (vehicleID, make, model, year,
dailyRate, status, passengerCapacity, engineCapacity) VALUES (%s, %s, %s,
%s, %s, %s, %s, %s)"
        try:
            cursor = connection.cursor()
            cursor.execute(sql, (vehicle.id, vehicle.make, vehicle.model,
vehicle.year, vehicle.daily_rate, vehicle.status,
vehicle.passenger_capacity, vehicle.engine_capacity))
            connection.commit()
            return cursor.lastrowid
        except Exception as e:
            raise e
        finally:
            if connection.is_connected():
                cursor.close()
                self.db_util.close_connection(connection)

    def removeCar(self, carID: int):
        connection = self.db_util.get_connection()
```

```python
        sql = "DELETE FROM Vehicle WHERE vehicleID = %s"
        try:
            cursor = connection.cursor()
            cursor.execute(sql, (carID,))
            connection.commit()
            if cursor.rowcount == 0:
                raise CarNotFoundException("Car with ID {} not
found".format(carID))
            print("Car removed successfully!")
        except Exception as e:
            raise e
        finally:
            if connection.is_connected():
                cursor.close()
                self.db_util.close_connection(connection)

    def listAvailableCars(self) -> List[Vehicle]:
        connection = self.db_util.get_connection()
        sql = "SELECT * FROM Vehicle WHERE status = 'available'"
        try:
            cursor = connection.cursor()
            cursor.execute(sql)
            results = cursor.fetchall()
            vehicles = [Vehicle(*row) for row in results]
            return vehicles
        except Exception as e:
            raise e
        finally:
            if connection.is_connected():
                cursor.close()
                self.db_util.close_connection(connection)

    def listRentedCars(self) -> List[Vehicle]:
        connection = self.db_util.get_connection()
        sql = "SELECT * FROM Vehicle WHERE status = 'notAvailable'"
        try:
            cursor = connection.cursor()
            cursor.execute(sql)
            results = cursor.fetchall()
            vehicles = [Vehicle(*row) for row in results]
            return vehicles
        except Exception as e:
            raise e
        finally:
            if connection.is_connected():
                cursor.close()
                self.db_util.close_connection(connection)

    def findCarById(self, carID: int) -> Vehicle:
        connection = self.db_util.get_connection()
        sql = "SELECT * FROM Vehicle WHERE vehicleID = %s"
        try:
            cursor = connection.cursor()
            cursor.execute(sql, (carID,))
            result = cursor.fetchone()
            if result is None:
                raise CarNotFoundException("Car with ID {} not
found".format(carID))
            return Vehicle(*result)
        except Exception as e:
            raise e
        finally:
            if connection.is_connected():
                cursor.close()
                self.db_util.close_connection(connection)

    def addCustomer(self, customer: Customer):
        connection = self.db_util.get_connection()
        sql = "INSERT INTO Customer (customerID, firstName, lastName,
email, phoneNumber) VALUES (%s, %s, %s, %s, %s)"
        try:
            cursor = connection.cursor()
            cursor.execute(sql, (customer.customerID, customer.firstName,
customer.lastName, customer.email, customer.phoneNumber))
            connection.commit()
        except Exception as e:
            raise e
        finally:
            if connection.is_connected():
                cursor.close()
                self.db_util.close_connection(connection)

    def removeCustomer(self, customerID: int):
        connection = self.db_util.get_connection()
        sql = "DELETE FROM Customer WHERE customerID = %s"
        try:
            cursor = connection.cursor()
            cursor.execute(sql, (customerID,))
            connection.commit()
            if cursor.rowcount == 0:
                raise CustomerNotFoundException("Customer with ID {} not
found".format(customerID))
            print("Customer removed successfully!")
        except Exception as e:
            raise e
        finally:
            if connection.is_connected():
                cursor.close()
                self.db_util.close_connection(connection)

    def listCustomers(self) -> List[Customer]:
        connection = self.db_util.get_connection()
        sql = "SELECT * FROM Customer"
        try:
            cursor = connection.cursor()
            cursor.execute(sql)
            results = cursor.fetchall()
            customers = [Customer(*row) for row in results]
            return customers
        except Exception as e:
            raise e
        finally:
            if connection.is_connected():
                cursor.close()
                self.db_util.close_connection(connection)

    def findCustomerById(self, customerID: int) -> Customer:
        connection = self.db_util.get_connection()
        sql = "SELECT * FROM Customer WHERE customerID = %s"
        try:
            cursor = connection.cursor()
            cursor.execute(sql, (customerID,))
            result = cursor.fetchone()
            if result is None:
                raise CustomerNotFoundException("Customer with ID {} not
found".format(customerID))
            return Customer(*result)
        except Exception as e:
            raise e
        finally:
            if connection is not None and connection.is_connected():
                cursor.close()
                self.db_util.close_connection(connection)

    def createLease(self, lease):
        connection = self.db_util.get_connection()
        sql = "INSERT INTO Lease (leaseID, vehicleID, customerID,
startDate, endDate, type) VALUES (%s, %s, %s, %s, %s, %s)"
        try:
            cursor = connection.cursor()
            cursor.execute(sql, (lease.leaseID, lease.vehicleID,
lease.customerID, lease.startDate, lease.endDate, lease.type))
            connection.commit()
            lease.leaseID = cursor.lastrowid
            return lease
        except Exception as e:
            raise e
        finally:
            if connection.is_connected():
                cursor.close()
                self.db_util.close_connection(connection)

    def returnCar(self, leaseID: int) -> Lease:
        connection = self.db_util.get_connection()
        sql = "UPDATE Lease SET endDate = CURRENT_DATE WHERE leaseID = %s
AND endDate > CURRENT_DATE"
        try:
            cursor = connection.cursor()
            cursor.execute(sql, (leaseID,))
            connection.commit()
            if cursor.rowcount == 0:
                raise LeaseNotFoundException("Lease with ID {} not found or
already ended".format(leaseID))
            return self.findLeaseById(leaseID)
        except Exception as e:
            raise e
        finally:
            if connection.is_connected():
                cursor.close()
                self.db_util.close_connection(connection)

    def listActiveLeases(self) -> List[Lease]:
        connection = self.db_util.get_connection()
        sql = "SELECT * FROM Lease WHERE endDate > CURRENT_DATE"
        try:
            cursor = connection.cursor()
            cursor.execute(sql)
            results = cursor.fetchall()
            leases = [Lease(*row) for row in results]
            return leases
        except Exception as e:
            raise e
        finally:
            if connection.is_connected():
                cursor.close()
                self.db_util.close_connection(connection)

    def listLeaseHistory(self) -> List[Lease]:
        connection = self.db_util.get_connection()
        sql = "SELECT * FROM Lease WHERE endDate <= CURRENT_DATE"
        try:
            cursor = connection.cursor()
            cursor.execute(sql)
            results = cursor.fetchall()
            leases = [Lease(*row) for row in results]
            return leases
        except Exception as e:
            raise e
        finally:
            if connection.is_connected():
                cursor.close()
                self.db_util.close_connection(connection)

    def recordPayment(self, lease: Lease, amount: float) -> None:
        connection = self.db_util.get_connection()
        sql = "INSERT INTO Payment (leaseID, paymentDate, amount) VALUES
(%s, CURRENT_DATE, %s)"
        try:
            cursor = connection.cursor()
            cursor.execute(sql, (lease.leaseID, amount))
            connection.commit()
        except Exception as e:
            raise e
        finally:
            if connection.is_connected():
                cursor.close()
                self.db_util.close_connection(connection)

    def findLeaseById(self, leaseID: int) -> Lease:
        connection = self.db_util.get_connection()
        sql = "SELECT * FROM Lease WHERE leaseID = %s"
        try:
            cursor = connection.cursor()
            cursor.execute(sql, (leaseID,))
            result = cursor.fetchone()
            if result is None:
                raise LeaseNotFoundException("Lease with ID {} not
found".format(leaseID))
            return Lease(*result)
        except Exception as e:
            raise e
        finally:
            if connection.is_connected():
                cursor.close()
                self.db_util.close_connection(connection)
```

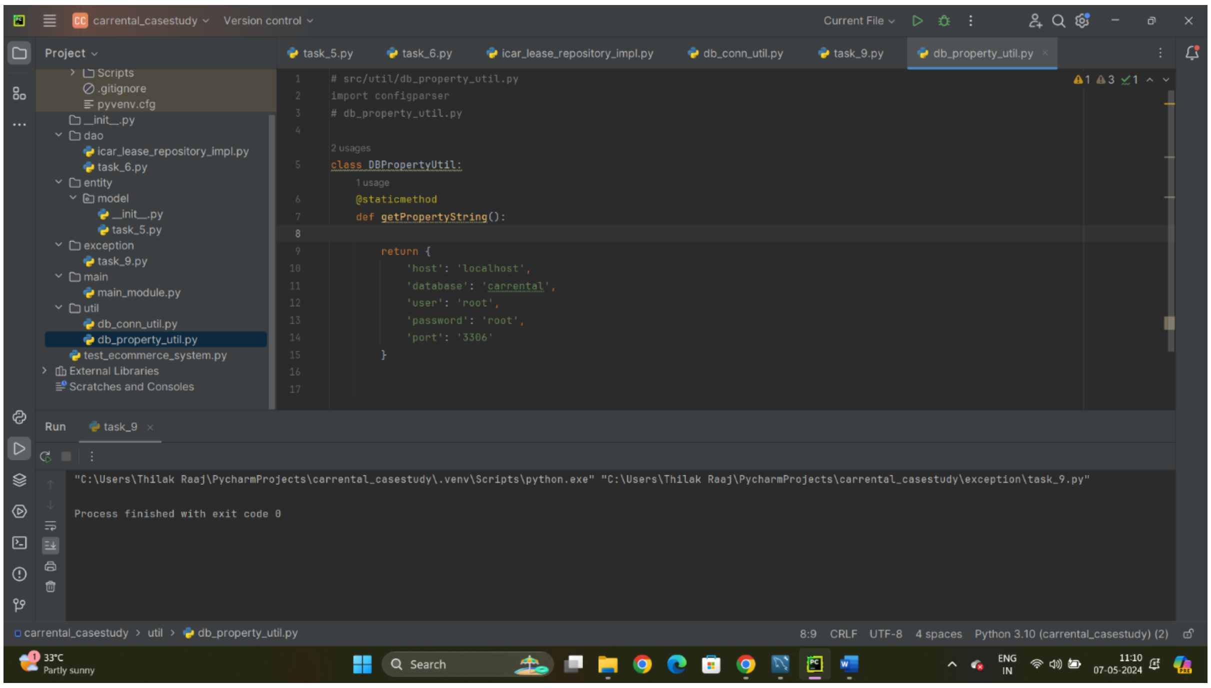


```python
class ICarLeaseRepositoryImpl(ICarLeaseRepository):
    def findLeaseById(self, leaseID: int) -> Lease:
        connection = self.db_util.get_connection()
        sql = "SELECT * FROM Lease WHERE leaseID = %s"
        try:
            cursor = connection.cursor()
            cursor.execute(sql, (leaseID,))
            result = cursor.fetchone()
            if result is None:
                raise LeaseNotFoundException("Lease with ID {} not found".format(leaseID))
            return Lease(*result)
        except Exception as e:
            raise e
        finally:
            if connection.is_connected():
                cursor.close()
                self.db_util.close_connection(connection)
```

```
"C:\Users\Thilak Raaj\PycharmProjects\carrental_casestudy\.venv\Scripts\python.exe" "C:\Users\Thilak Raaj\PycharmProjects\carrental_casestudy\dao\icar_lease_repository_impl.py'

Process finished with exit code 0
```

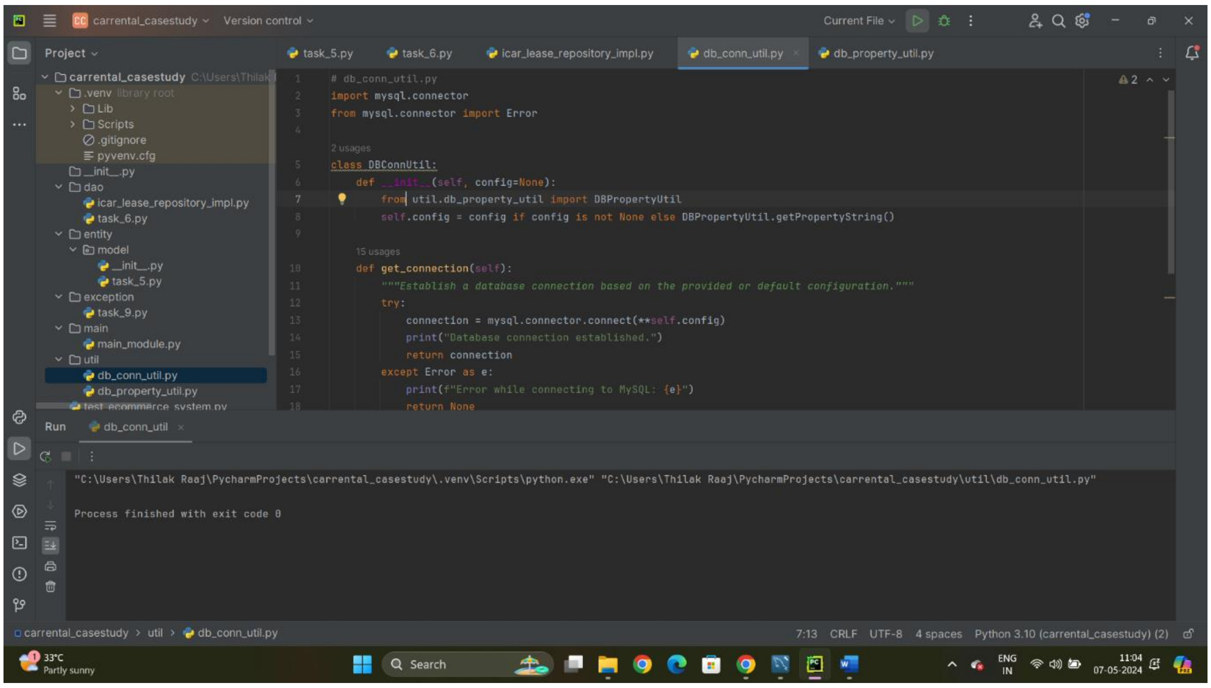
8. Connect your application to the SQL database and write code to establish a connection to your SQL database. • Create a utility class DBConnection in a package util with a static variable connection of Type Connection and a static method getConnection() which returns connection. • Connection properties supplied in the connection string should be read from a property file. • Create a utility class PropertyUtil which contains a static method named getPropertyString() which reads a property fie containing connection details like hostname, dbname, username, password, port number and returns a connection string.

```python
class DBPropertyUtil:
    @staticmethod
    def getPropertyString():

        return {
            'host': 'localhost',
            'database': 'carrental',
            'user': 'root',
            'password': 'root',
            'port': '3306'
        }

# db_conn_util.py
import mysql.connector
from mysql.connector import Error

class DBConnUtil:
    def __init__(self, config=None):
        from util.db_property_util import DBPropertyUtil
        self.config = config if config is not None else
DBPropertyUtil.getPropertyString()

    def get_connection(self):
        """Establish a database connection based on the provided or default
configuration."""
        try:
            connection = mysql.connector.connect(**self.config)
            print("Database connection established.")
            return connection
        except Error as e:
            print(f"Error while connecting to MySQL: {e}")
            return None

    def close_connection(self, connection):
        """Close the provided database connection."""
        if connection:
            if connection.is_connected():
                connection.close()
                print("Database connection closed.")
```



```python
# src/util/db_property_util.py
import configparser
# db_property_util.py

class DBPropertyUtil:
    @staticmethod
    def getPropertyString():

        return {
            'host': 'localhost',
            'database': 'carrental',
            'user': 'root',
            'password': 'root',
            'port': '3306'
        }
```

```
"C:\Users\Thilak Raaj\PycharmProjects\carrental_casestudy\.venv\Scripts\python.exe" "C:\Users\Thilak Raaj\PycharmProjects\carrental_casestudy\exception\task_9.py"

Process finished with exit code 0
```

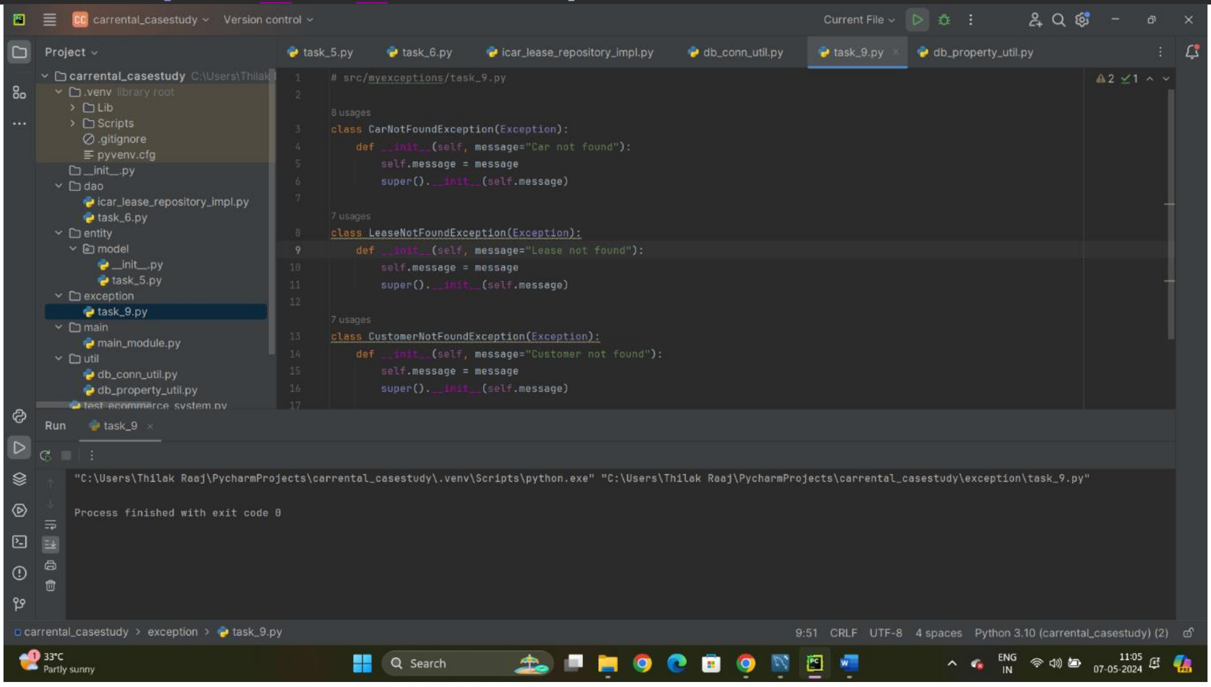


```python
# db_conn_util.py
import mysql.connector
from mysql.connector import Error

class DBConnUtil:
    def __init__(self, config=None):
        from util.db_property_util import DBPropertyUtil
        self.config = config if config is not None else DBPropertyUtil.getPropertyString()

    def get_connection(self):
        """Establish a database connection based on the provided or default configuration."""
        try:
            connection = mysql.connector.connect(**self.config)
            print("Database connection established.")
            return connection
        except Error as e:
            print(f"Error while connecting to MySQL: {e}")
            return None
```

```
"C:\Users\Thilak Raaj\PycharmProjects\carrental_casestudy\.venv\Scripts\python.exe" "C:\Users\Thilak Raaj\PycharmProjects\carrental_casestudy\util\db_conn_util.py"

Process finished with exit code 0
```

9. Create the exceptions in package myexceptions and create the following custom exceptions and throw them in methods whenever needed. Handle all the exceptions in main method, • CarNotFoundException: throw this exception when user enters an invalid car id which doesn't exist in db. • LeaseNotFoundException: throw this exception when user enters an invalid lease id which doesn't exist in db. • CustomerrNotFoundException: throw this exception when user enters an invalid customer id which doesn't exist in db.

```python
# src/myexceptions/task_9.py

class CarNotFoundException(Exception):
    def __init__(self, message="Car not found"):
        self.message = message
        super().__init__(self.message)

class LeaseNotFoundException(Exception):
    def __init__(self, message="Lease not found"):
        self.message = message
        super().__init__(self.message)

class CustomerNotFoundException(Exception):
    def __init__(self, message="Customer not found"):
        self.message = message
        super().__init__(self.message)
```



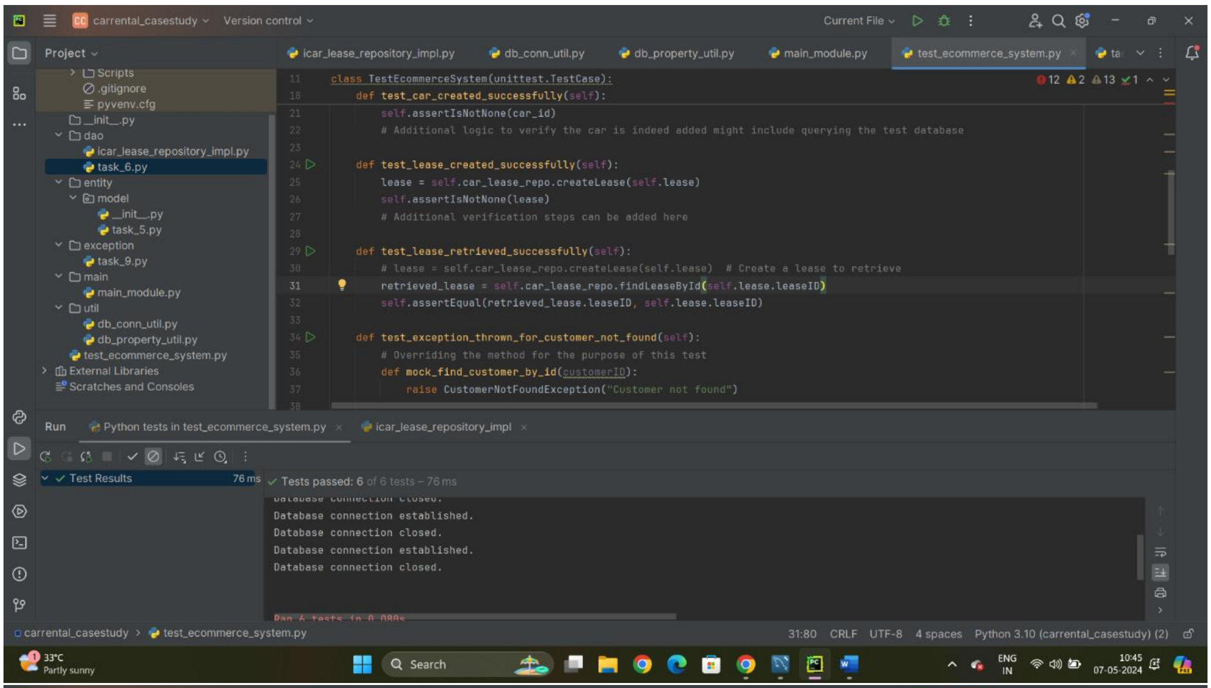
```python
# src/myexceptions/task_9.py

class CarNotFoundException(Exception):
    def __init__(self, message="Car not found"):
        self.message = message
        super().__init__(self.message)

class LeaseNotFoundException(Exception):
    def __init__(self, message="Lease not found"):
        self.message = message
        super().__init__(self.message)

class CustomerNotFoundException(Exception):
    def __init__(self, message="Customer not found"):
        self.message = message
        super().__init__(self.message)
```

```
"C:\Users\Thilak Raaj\PycharmProjects\carrental_casestudy\.venv\Scripts\python.exe" "C:\Users\Thilak Raaj\PycharmProjects\carrental_casestudy\exception\task_9.py"

Process finished with exit code 0
```

Unit Testing: 10. Create Unit test cases for Ecommerce System are essential to ensure the correctness and reliability of your system. Following questions to guide the creation of Unit test cases: • Write test case to test car created successfully or not. • Write test case to test lease is created successfully or not. • Write test case to test lease is retrieved successfully or not. • write test case to test exception is thrown correctly or not when customer id or car id or lease id not found in database.

```python
import unittest
from unittest.mock import MagicMock, patch
from entity.model.task_5 import Vehicle, Lease
```

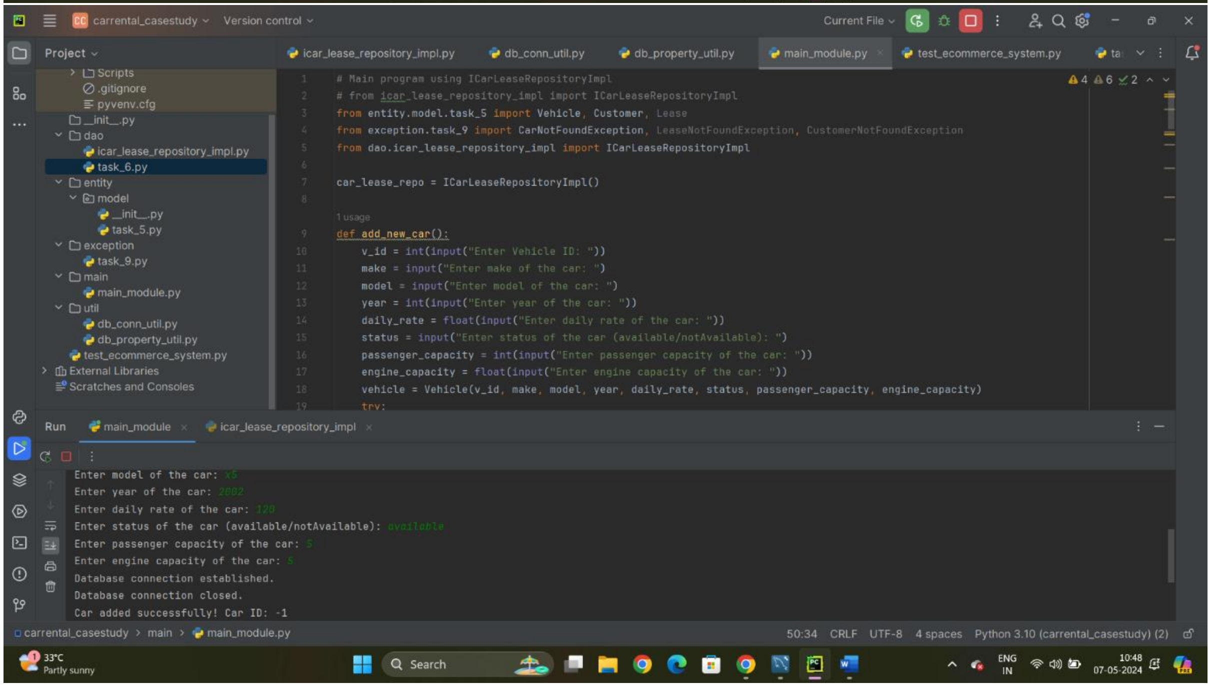
```python
                                 # YourServiceClass, CarNotFoundException,
LeaseNotFoundException, CustomerNotFoundException)

import unittest
from dao.icar_lease_repository_impl import ICarLeaseRepositoryImpl

from exception.task_9 import CarNotFoundException, LeaseNotFoundException,
CustomerNotFoundException

class TestEcommerceSystem(unittest.TestCase):
    def setUp(self):
        # Assuming the ICarLeaseRepositoryImpl can be adjusted to use a
test database or similar setup
        self.car_lease_repo = ICarLeaseRepositoryImpl()
        self.vehicle = Vehicle(3, "Camry", 2022, 50, 30,"available", 5, 2)
        self.lease = Lease(1, 3, 1, "2022-01-01", "2022-01-10",
"DailyLease")

    def test_car_created_successfully(self):
        # Directly insert and verify in the test database
        car_id = self.car_lease_repo.addCar(self.vehicle)
        self.assertIsNotNone(car_id)

    def test_lease_created_successfully(self):
        lease = self.car_lease_repo.createLease(self.lease)
        self.assertIsNotNone(lease)

    def test_lease_retrieved_successfully(self):
        retrieved_lease =
self.car_lease_repo.findLeaseById(self.lease.leaseID)
        self.assertEqual(retrieved_lease.leaseID, self.lease.leaseID)

    def test_exception_thrown_for_customer_not_found(self):
        def mock_find_customer_by_id(customerID):
            raise CustomerNotFoundException("Customer not found")

        self.car_lease_repo.findCustomerById = mock_find_customer_by_id

        with self.assertRaises(CustomerNotFoundException):
            self.car_lease_repo.findCustomerById(999)

    def test_exception_thrown_for_car_not_found(self):
        def mock_find_car_by_id(carID):
            raise CarNotFoundException("Car not found")

        self.car_lease_repo.findCarById = mock_find_car_by_id

        with self.assertRaises(CarNotFoundException):
            self.car_lease_repo.findCarById(999)

    def test_exception_thrown_for_lease_not_found(self):
        def mock_find_lease_by_id(leaseID):
            raise LeaseNotFoundException("Lease not found")

        self.car_lease_repo.findLeaseById = mock_find_lease_by_id

        with self.assertRaises(LeaseNotFoundException):
            self.car_lease_repo.findLeaseById(999)

if __name__ == '__main__':
    unittest.main()
```



```python
class TestEcommerceSystem(unittest.TestCase):
    def test_car_created_successfully(self):
        self.assertIsNotNone(car_id)
        # Additional logic to verify the car is indeed added might include querying the test database

    def test_lease_created_successfully(self):
        lease = self.car_lease_repo.createLease(self.lease)
        self.assertIsNotNone(lease)
        # Additional verification steps can be added here

    def test_lease_retrieved_successfully(self):
        # lease = self.car_lease_repo.createLease(self.lease)  # Create a lease to retrieve
        retrieved_lease = self.car_lease_repo.findLeaseById(self.lease.leaseID)
        self.assertEqual(retrieved_lease.leaseID, self.lease.leaseID)

    def test_exception_thrown_for_customer_not_found(self):
        # Overriding the method for the purpose of this test
        def mock_find_customer_by_id(customerID):
            raise CustomerNotFoundException("Customer not found")
```

✓ Tests passed: 6 of 6 tests – 76 ms

```
Database connection established.
Database connection closed.
Database connection established.
Database connection closed.


Ran 6 tests in 0.080s
```



```python
# Main program using ICarLeaseRepositoryImpl
# from icar_lease_repository_impl import ICarLeaseRepositoryImpl
from entity.model.task_5 import Vehicle, Customer, Lease
from exception.task_9 import CarNotFoundException, LeaseNotFoundException, CustomerNotFoundException
from dao.icar_lease_repository_impl import ICarLeaseRepositoryImpl

car_lease_repo = ICarLeaseRepositoryImpl()

1 usage
def add_new_car():
    v_id = int(input("Enter Vehicle ID: "))
    make = input("Enter make of the car: ")
    model = input("Enter model of the car: ")
    year = int(input("Enter year of the car: "))
    daily_rate = float(input("Enter daily rate of the car: "))
    status = input("Enter status of the car (available/notAvailable): ")
    passenger_capacity = int(input("Enter passenger capacity of the car: "))
    engine_capacity = float(input("Enter engine capacity of the car: "))
    vehicle = Vehicle(v_id, make, model, year, daily_rate, status, passenger_capacity, engine_capacity)
    try:
```

```
Enter model of the car: x5
Enter year of the car: 2002
Enter daily rate of the car: 120
Enter status of the car (available/notAvailable): available
Enter passenger capacity of the car: 5
Enter engine capacity of the car: 5
Database connection established.
Database connection closed.
Car added successfully! Car ID: -1
```