Portfolio Assignment for

Master of Science in Information Technology

Software Design and Programming

James McKenna

University of Denver College of Professional Studies

11/14/2025

Faculty: Nirav Shah, M.S.

Director: Cathie Wilson, M.S.

Dean: Michael J. McGuire, MLS

Abstract

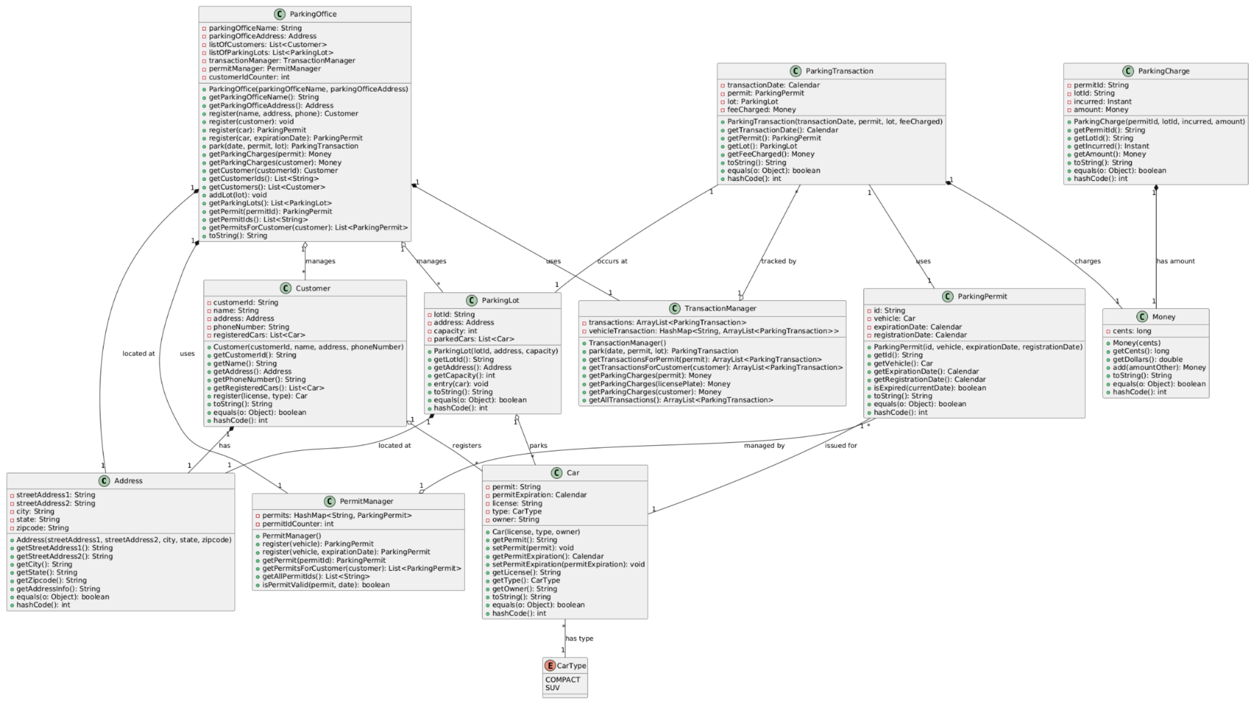
Our Portfolio Assignment is the final assignment for our Object-Oriented Methods and Programming 1. In this final assignment, creations of four new classes were made in order to create a Transaction Manager, Permit Manager, Parking Permit, and Parking Transaction class in order to fully facilitate and bring the whole parking system together. This also tied up any remaining classes that needed to either be created or fixed in previous assignments. Overall this was a tough and interesting assignment as these final classes directly tied into all our previous classes as a whole.

CONTENTS

Introduction

With this particular assignment I found myself going back and redeveloping a lot of code. There were a lot of classes where I had to not only add the new equal and hashCode methods, but I also had to include more getters so that there was less coupling between classes. By using more getters I was able to get rid of tighter coupling with classes so that only by calling one part of a getter one class would be able to pull all it needed and was less dependent on the classes themselves. I had to also incorporate the new frameworks for collections such as our equals and hashCode. Also by forcing encapsulation now I was able to make it so that whoever is trying to utilize or incorporate our code has to do it through my getters and can't manipulate our data directly.

Class Diagram


ParkingOffice
parkingOfficeName: String
parkingOfficeAddress: Address
listOfCustomers: List<Customer>
listOfParkingLots: List<ParkingLot>
transactionManager: TransactionManager
permitManager: PermitManager
customerIdCounter: int
ParkingOffice(parkingOfficeName, parkingOfficeAddress)
getParkingOfficeName(): String
getParkingOfficeAddress(): Address
register(name, address, phone): Customer
register(customer): void
register(car): ParkingPermit
register(car, expirationDate): ParkingPermit
park(date, permit, lot): ParkingTransaction
getParkingCharges(permit): Money
getParkingCharges(customer): Money
getCustomer(customerId): Customer
getCustomerIds(): List<String>
getCustomers(): List<Customer>
addLot(lot): void
getParkingLots(): List<ParkingLot>
getPermit(permitId): ParkingPermit
getPermitIds(): List<String>
getPermitsForCustomer(customer): List<ParkingPermit>
toString(): String
ParkingTransaction
transactionDate: Calendar
permit: ParkingPermit
lot: ParkingLot
feeCharged: Money
ParkingTransaction(transactionDate, permit, lot, feeCharged)
getTransactionDate(): Calendar
getPermit(): ParkingPermit
getLot(): ParkingLot
getFeeCharged(): Money
toString(): String
equals(o: Object): boolean
hashCode(): int
ParkingCharge
permitId: String
lotId: String
incurred: Instant
amount: Money
ParkingCharge(permitId, lotId, incurred, amount)
getPermitId(): String
getLotId(): String
getIncurred(): Instant
getAmount(): Money
toString(): String
equals(o: Object): boolean
hashCode(): int
manages
manages
uses
occurs at
tracked by
uses
charges
has amount
Customer
customerId: String
name: String
address: Address
phoneNumber: String
registeredCars: List<Car>
Customer(customerId, name, address, phoneNumber)
getCustomerId(): String
getName(): String
getAddress(): Address
getPhoneNumber(): String
getRegisteredCars(): List<Car>
register(license, type): Car
toString(): String
equals(o: Object): boolean
hashCode(): int
ParkingLot
lotId: String
address: Address
capacity: int
parkedCars: List<Car>
ParkingLot(lotId, address, capacity)
getLotId(): String
getAddress(): Address
getCapacity(): int
entry(car): void
toString(): String
equals(o: Object): boolean
hashCode(): int
TransactionManager
transactions: ArrayList<ParkingTransaction>
vehicleTransaction: HashMap<String, ArrayList<ParkingTransaction>>
TransactionManager()
park(date, permit, lot): ParkingTransaction
getTransactionsForPermit(permit): ArrayList<ParkingTransaction>
getTransactionsForCustomer(customer): ArrayList<ParkingTransaction>
getParkingCharges(permit): Money
getParkingCharges(licensePlate): Money
getParkingCharges(customer): Money
getAllTransactions(): ArrayList<ParkingTransaction>
ParkingPermit
id: String
vehicle: Car
expirationDate: Calendar
registrationDate: Calendar
ParkingPermit(id, vehicle, expirationDate, registrationDate)
getId(): String
getVehicle(): Car
getExpirationDate(): Calendar
getRegistrationDate(): Calendar
isExpired(currentDate): boolean
toString(): String
equals(o: Object): boolean
hashCode(): int
Money
cents: long
Money(cents)
getCents(): long
getDollars(): double
add(amountOther): Money
toString(): String
equals(o: Object): boolean
hashCode(): int
located at
uses
has
located at
registers
parks
managed by
issued for
Address
streetAddress1: String
streetAddress2: String
city: String
state: String
zipcode: String
Address(streetAddress1, streetAddress2, city, state, zipcode)
getStreetAddress1(): String
getStreetAddress2(): String
getCity(): String
getState(): String
getZipcode(): String
getAddressInfo(): String
equals(o: Object): boolean
hashCode(): int
PermitManager
permits: HashMap<String, ParkingPermit>
permitIdCounter: int
PermitManager()
register(vehicle): ParkingPermit
register(vehicle, expirationDate): ParkingPermit
getPermit(permitId): ParkingPermit
getPermitsForCustomer(customer): List<ParkingPermit>
getAllPermitIds(): List<String>
isPermitValid(permit, date): boolean
Car
permit: String
permitExpiration: Calendar
license: String
type: CarType
owner: String
Car(license, type, owner)
getPermit(): String
setPermit(permit): void
getPermitExpiration(): Calendar
setPermitExpiration(permitExpiration): void
getLicense(): String
getType(): CarType
getOwner(): String
toString(): String
equals(o: Object): boolean
hashCode(): int
has type
CarType
COMPACT
SUV

This class diagram is an accumulation of all the previous class diagrams that has been updated and summed into one UML Diagram. The Parking System now manages permits, lots, and some parking fees and bills. First is the Parking Office which handles registry, parking charges, adding customers, adding lots and fetching permits. Next would be the Parking Transaction class which is responsible for lots, fees, money, and transaction details, next is the parking charge where this will handle the amounts, Ids, and conversion. The Parking Office overall uses the Transaction Manager, it manages the Parking Lot class and the Customer Class; while also has a permit manager associated with it. The Customer Class has addresses associated with it and also registers cars. Finally we have our Parking Permit class which is managed by our Permit Manager and issues our Cars as well. Overall everything is very coupled and tied nicely together for this final assignment where certain classes are managing or using other classes, while some of our top classes are in charge of managing or tracking other classes we've developed in this use case.

What did I learn from this project?

What I learned overall in this project was primarily Java. I wasn't super familiar with Java before taking this class as I was more used to Python and C++ from my professional experience. However I can see that Java is a nice middle ground in between Python and C++ where its a high level language just like the other two, but it has more control and use cases aside from Python. I also was very happy to build a parking management system and learn how to tie this in with other classes. My only big takeaway from this whole project was that I did wish there were some easier tie-ins between classes and assignments. A lot of the class built up in between each assignment required me to use a lot of assumptions even with some of the guidance we had from our class assignments.

New tools and designs

For new tools and design the biggest takeaways from this class was using Intellij and using PlantUML.com in order to make UML Diagrams for this class. I've never fully delved into UML diagrams too much at work but it does make understanding architecture and software so much easier. The UML Diagrams were also very easy for me to update once I had a base layer as this allowed me to keep making changes and track the changes that I had in between revisions. Other than that the biggest feature I also was using was junit tests. Doing unit testing is a big takeaway at every job I've been at but each unit testing system is substantially different. Using junit I was able to not only take away the experience but also understand how to properly test and develop my code along the way.

What had I wished I done differently?

Personally I wish I knew a lot of the architecture or main management classes I would've needed earlier on. Adding in and incorporating new classes and new updates into our code was harder as this system started to grow. It also made it so that my classes and architecture had to constantly be retested or new unit tests would have to be developed due to refactoring classes as the system needed more classes or more unit testing. I think if a system design approach was started first, or a software design pattern was applied to our system it would’ve made this assignment a lot easier in hindsight.

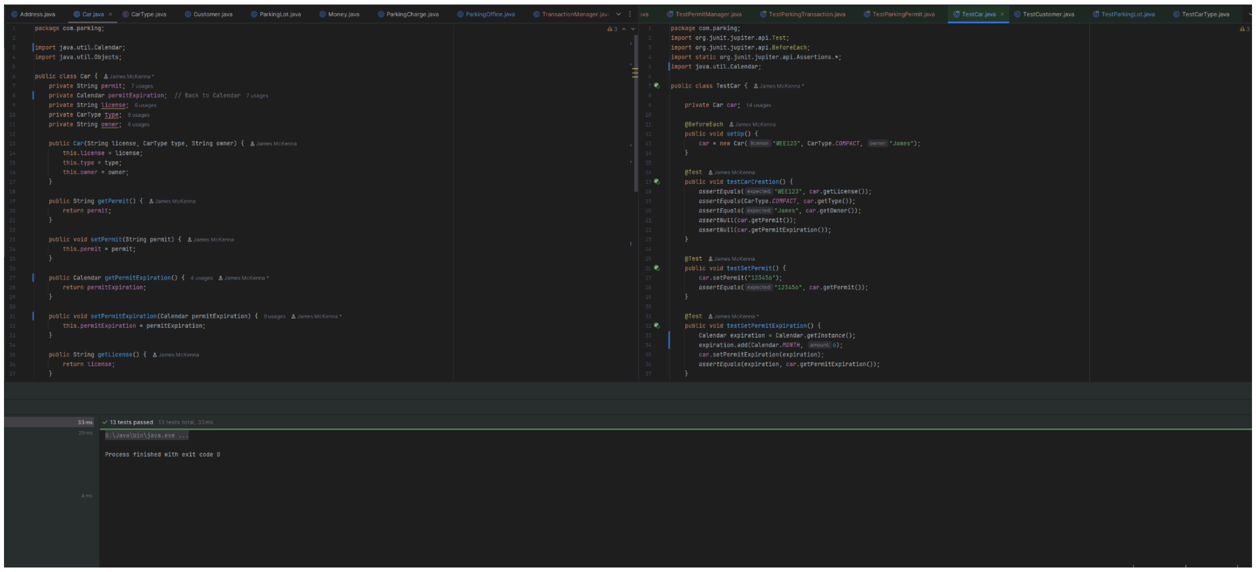
If I had more time what enhancements would I make?

Kind of what I hinted at before but I believe that adding in other classes that could be truncated into one another, or having fewer classes and one that handled more would’ve been a huge take away. A lot of the classes that were designed either did very minimal work or they interfaced with fewer classes that rolling them into the bigger ones might’ve helped. I know this would’ve increased dependencies on the class that started to grow but managing so many classes started to become harder and harder as the system grew. For the system itself I think having timers for parking spots, individual parking spots, payment methods, events, security or access control, reporting systems, or even a violation tracking update would all be future enhancements that could be made to our parking system. I think also making the system a little easier to understand might also help.

Successful Unit Test Screenshots

For this final assignment I only included the classes that were either added in or need a new unit test for updates. I personally didn’t add in every single unit test being successfully run since there were only two classes from previous assignments that needed to be updated such as my Car and Parking Office classes. The rest of the unit tests below were screenshots that were captured from the four new classes that needed to be added such as the Parking Permit, Parking Transaction, Permit Manager and Transaction Manager classes.

TestCar.java

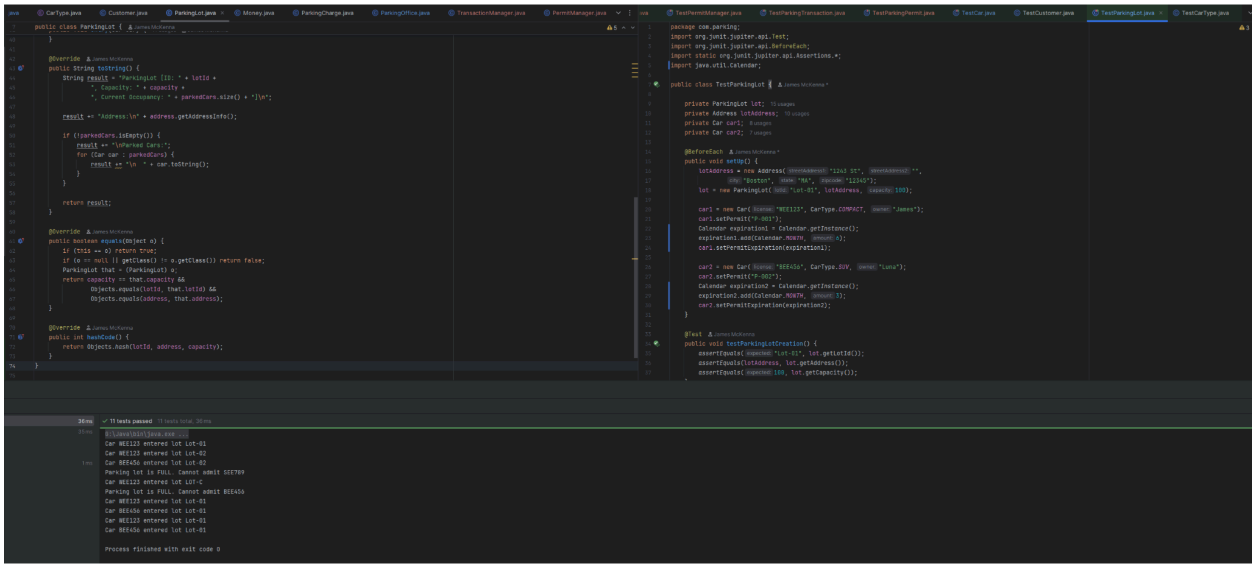


```java
package com.parking;

import java.util.Calendar;
import java.util.Objects;

public class Car {
    private String permit;
    private Calendar permitExpiration;  // Back to Calendar
    private String license;
    private CarType type;
    private String owner;

    public Car(String license, CarType type, String owner) {
        this.license = license;
        this.type = type;
        this.owner = owner;
    }

    public String getPermit() {
        return permit;
    }

    public void setPermit(String permit) {
        this.permit = permit;
    }

    public Calendar getPermitExpiration() {
        return permitExpiration;
    }

    public void setPermitExpiration(Calendar permitExpiration) {
        this.permitExpiration = permitExpiration;
    }

    public String getLicense() {
        return license;
    }
```

```java
package com.parking;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import static org.junit.jupiter.api.Assertions.*;
import java.util.Calendar;

public class TestCar {

    private Car car;

    @BeforeEach
    public void setUp() {
        car = new Car("WEE123", CarType.COMPACT, "James");
    }

    @Test
    public void testCarCreation() {
        assertEquals("WEE123", car.getLicense());
        assertEquals(CarType.COMPACT, car.getType());
        assertEquals("James", car.getOwner());
        assertNull(car.getPermit());
        assertNull(car.getPermitExpiration());
    }

    @Test
    public void testSetPermit() {
        car.setPermit("123456");
        assertEquals("123456", car.getPermit());
    }

    @Test
    public void testSetPermitExpiration() {
        Calendar expiration = Calendar.getInstance();
        expiration.add(Calendar.MONTH, 6);
        car.setPermitExpiration(expiration);
        assertEquals(expiration, car.getPermitExpiration());
    }
```

```
✓ 13 tests passed  13 tests total, 33 ms
G:\Java\bin\java.exe ...

Process finished with exit code 0
```

TestParkingLot.java

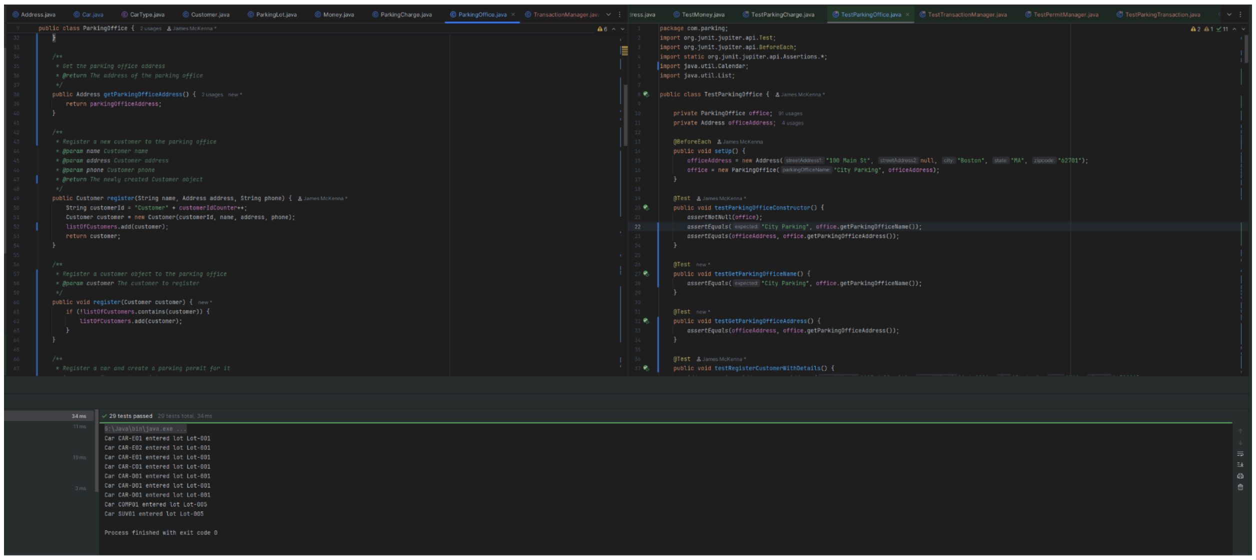


```java
public class ParkingLot {
    }

    @Override
    public String toString() {
        String result = "ParkingLot [ID: " + lotId +
                ", Capacity: " + capacity +
                ", Current Occupancy: " + parkedCars.size() + "]\n";

        result += "Address:\n" + address.getAddressInfo();

        if (!parkedCars.isEmpty()) {
            result += "\nParked Cars:";
            for (Car car : parkedCars) {
                result += "\n  " + car.toString();
            }
        }

        return result;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        ParkingLot that = (ParkingLot) o;
        return capacity == that.capacity &&
                Objects.equals(lotId, that.lotId) &&
                Objects.equals(address, that.address);
    }

    @Override
    public int hashCode() {
        return Objects.hash(lotId, address, capacity);
    }
}
```

```java
package com.parking;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import static org.junit.jupiter.api.Assertions.*;
import java.util.Calendar;

public class TestParkingLot {

    private ParkingLot lot;
    private Address lotAddress;
    private Car car1;
    private Car car2;

    @BeforeEach
    public void setUp() {
        lotAddress = new Address("1243 St", "",
                "Boston", "MA", "12345");
        lot = new ParkingLot("Lot-01", lotAddress, 100);

        car1 = new Car("WEE123", CarType.COMPACT, "James");
        car1.setPermit("P-001");
        Calendar expiration1 = Calendar.getInstance();
        expiration1.add(Calendar.MONTH, 6);
        car1.setPermitExpiration(expiration1);

        car2 = new Car("BEE456", CarType.SUV, "Luna");
        car2.setPermit("P-002");
        Calendar expiration2 = Calendar.getInstance();
        expiration2.add(Calendar.MONTH, 3);
        car2.setPermitExpiration(expiration2);
    }

    @Test
    public void testParkingLotCreation() {
        assertEquals("Lot-01", lot.getLotId());
        assertEquals(lotAddress, lot.getAddress());
        assertEquals(100, lot.getCapacity());
```

```
✓ 11 tests passed  11 tests total, 36 ms
G:\Java\bin\java.exe ...
Car WEE123 entered lot Lot-01
Car WEE123 entered lot Lot-02
Car BEE456 entered lot Lot-02
Parking lot is FULL. Cannot admit SEE789
Car WEE123 entered lot LOT-C
Parking lot is FULL. Cannot admit BEE456
Car WEE123 entered lot Lot-01
Car BEE456 entered lot Lot-01
Car WEE123 entered lot Lot-01
Car BEE456 entered lot Lot-01

Process finished with exit code 0
```

TestParkingOffice.java

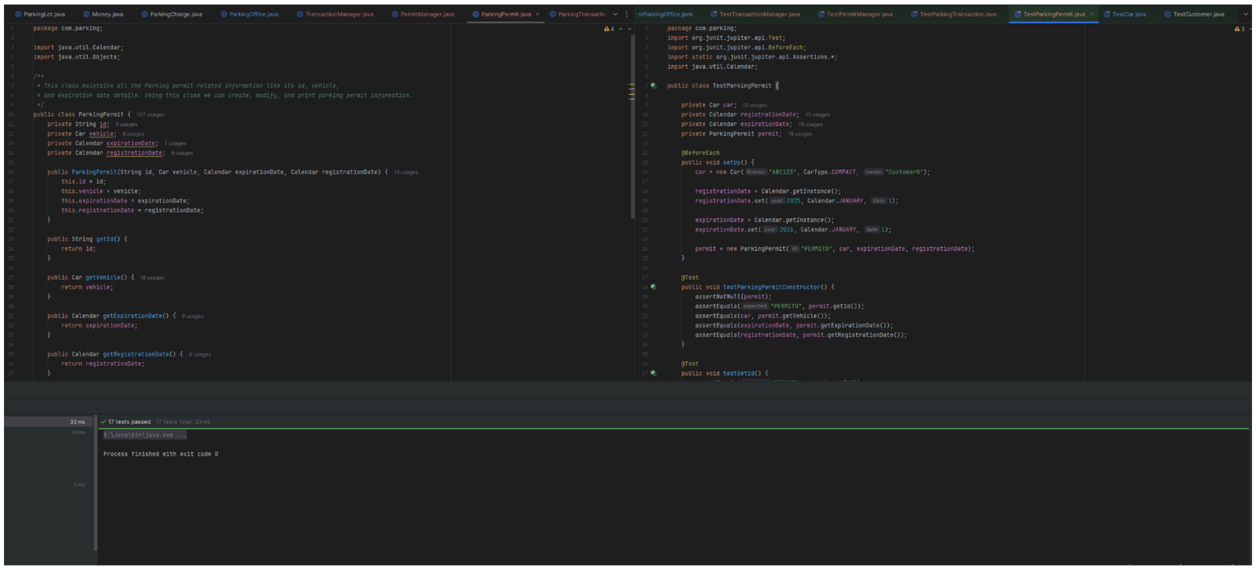


```java
public class ParkingOffice {
    }

    /**
     * Get the parking office address
     * @return The address of the parking office
     */
    public Address getParkingOfficeAddress() {
        return parkingOfficeAddress;
    }

    /**
     * Register a new customer to the parking office
     * @param name Customer name
     * @param address Customer address
     * @param phone Customer phone
     * @return The newly created Customer object
     */
    public Customer register(String name, Address address, String phone) {
        String customerId = "Customer" + customerIdCounter++;
        Customer customer = new Customer(customerId, name, address, phone);
        listOfCustomers.add(customer);
        return customer;
    }

    /**
     * Register a customer object to the parking office
     * @param customer The customer to register
     */
    public void register(Customer customer) {
        if (!listOfCustomers.contains(customer)) {
            listOfCustomers.add(customer);
        }
    }

    /**
     * Register a car and create a parking permit for it
```

```java
package com.parking;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import static org.junit.jupiter.api.Assertions.*;
import java.util.Calendar;
import java.util.List;

public class TestParkingOffice {

    private ParkingOffice office;
    private Address officeAddress;

    @BeforeEach
    public void setUp() {
        officeAddress = new Address("100 Main St", null, "Boston", "MA", "62701");
        office = new ParkingOffice("City Parking", officeAddress);
    }

    @Test
    public void testParkingOfficeConstructor() {
        assertNotNull(office);
        assertEquals("City Parking", office.getParkingOfficeName());
        assertEquals(officeAddress, office.getParkingOfficeAddress());
    }

    @Test
    public void testGetParkingOfficeName() {
        assertEquals("City Parking", office.getParkingOfficeName());
    }

    @Test
    public void testGetParkingOfficeAddress() {
        assertEquals(officeAddress, office.getParkingOfficeAddress());
    }

    @Test
    public void testRegisterCustomerWithDetails() {
```

```
✓ 29 tests passed  29 tests total, 34 ms
G:\Java\bin\java.exe ...
Car CAR-E01 entered lot Lot-001
Car CAR-E02 entered lot Lot-001
Car CAR-E01 entered lot Lot-001
Car CAR-C01 entered lot Lot-001
Car CAR-D01 entered lot Lot-001
Car CAR-D01 entered lot Lot-001
Car CAR-D01 entered lot Lot-001
Car COMP01 entered lot Lot-005
Car SUV01 entered lot Lot-005

Process finished with exit code 0
```

TestParkingPermit.java

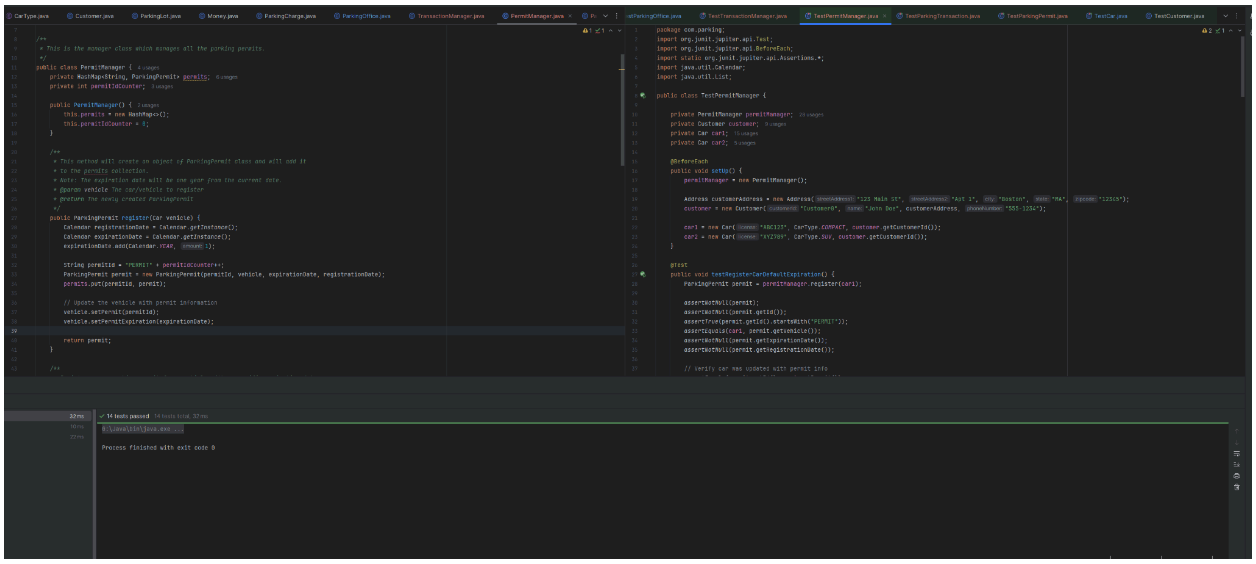


```java
package com.parking;

import java.util.Calendar;
import java.util.Objects;

/**
 * This class maintains all the Parking permit related information like its id, vehicle,
 * and expiration date details. Using this class we can create, modify, and print parking permit information.
 */
public class ParkingPermit {
    private String id;
    private Car vehicle;
    private Calendar expirationDate;
    private Calendar registrationDate;

    public ParkingPermit(String id, Car vehicle, Calendar expirationDate, Calendar registrationDate) {
        this.id = id;
        this.vehicle = vehicle;
        this.expirationDate = expirationDate;
        this.registrationDate = registrationDate;
    }

    public String getId() {
        return id;
    }

    public Car getVehicle() {
        return vehicle;
    }

    public Calendar getExpirationDate() {
        return expirationDate;
    }

    public Calendar getRegistrationDate() {
        return registrationDate;
    }
```

```java
package com.parking;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import static org.junit.jupiter.api.Assertions.*;
import java.util.Calendar;

public class TestParkingPermit {

    private Car car;
    private Calendar registrationDate;
    private Calendar expirationDate;
    private ParkingPermit permit;

    @BeforeEach
    public void setUp() {
        car = new Car("ABC123", CarType.COMPACT, "Customer0");

        registrationDate = Calendar.getInstance();
        registrationDate.set(2025, Calendar.JANUARY, 1);

        expirationDate = Calendar.getInstance();
        expirationDate.set(2026, Calendar.JANUARY, 1);

        permit = new ParkingPermit("PERMIT0", car, expirationDate, registrationDate);
    }

    @Test
    public void testParkingPermitConstructor() {
        assertNotNull(permit);
        assertEquals("PERMIT0", permit.getId());
        assertEquals(car, permit.getVehicle());
        assertEquals(expirationDate, permit.getExpirationDate());
        assertEquals(registrationDate, permit.getRegistrationDate());
    }

    @Test
    public void testGetId() {
```

```
✓ 17 tests passed  17 tests total, 33 ms
G:\Java\bin\java.exe ...

Process finished with exit code 0
```

TestPermitManager.java

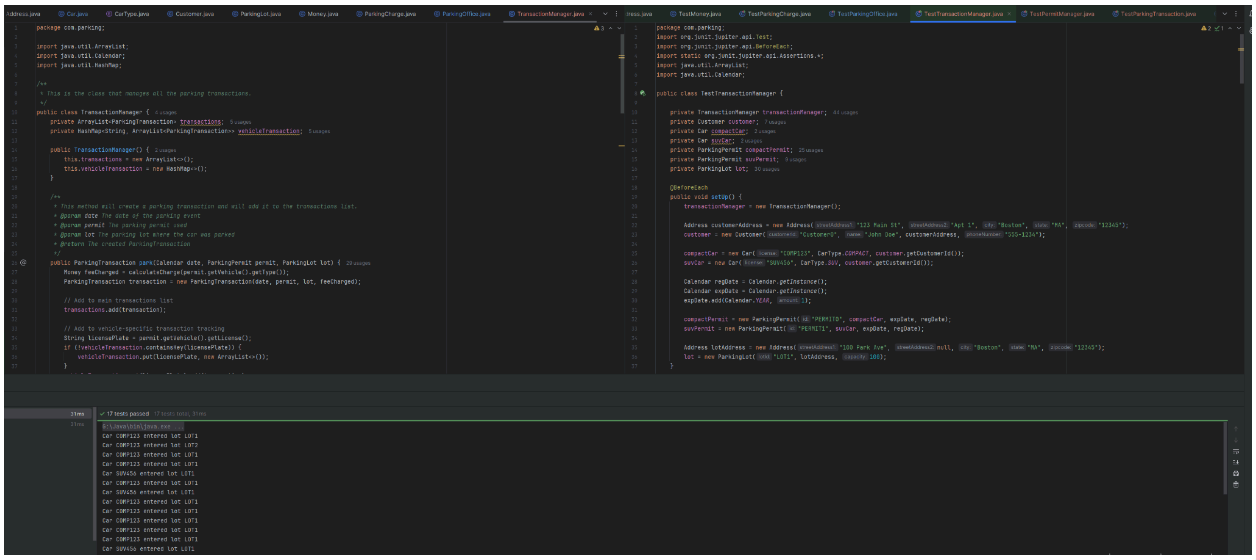


```java
/**
 * This is the manager class which manages all the parking permits.
 */
public class PermitManager {
    private HashMap<String, ParkingPermit> permits;
    private int permitIdCounter;

    public PermitManager() {
        this.permits = new HashMap<>();
        this.permitIdCounter = 0;
    }

    /**
     * This method will create an object of ParkingPermit class and will add it
     * to the permits collection.
     * Note: The expiration date will be one year from the current date.
     * @param vehicle The car/vehicle to register
     * @return The newly created ParkingPermit
     */
    public ParkingPermit register(Car vehicle) {
        Calendar registrationDate = Calendar.getInstance();
        Calendar expirationDate = Calendar.getInstance();
        expirationDate.add(Calendar.YEAR, 1);

        String permitId = "PERMIT" + permitIdCounter++;
        ParkingPermit permit = new ParkingPermit(permitId, vehicle, expirationDate, registrationDate);
        permits.put(permitId, permit);

        // Update the vehicle with permit information
        vehicle.setPermit(permitId);
        vehicle.setPermitExpiration(expirationDate);

        return permit;
    }

    /**
```

```java
package com.parking;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import static org.junit.jupiter.api.Assertions.*;
import java.util.Calendar;
import java.util.List;

public class TestPermitManager {

    private PermitManager permitManager;
    private Customer customer;
    private Car car1;
    private Car car2;

    @BeforeEach
    public void setUp() {
        permitManager = new PermitManager();

        Address customerAddress = new Address("123 Main St", "Apt 1", "Boston", "MA", "12345");
        customer = new Customer("Customer0", "John Doe", customerAddress, "555-1234");

        car1 = new Car("ABC123", CarType.COMPACT, customer.getCustomerId());
        car2 = new Car("XYZ789", CarType.SUV, customer.getCustomerId());
    }

    @Test
    public void testRegisterCarDefaultExpiration() {
        ParkingPermit permit = permitManager.register(car1);

        assertNotNull(permit);
        assertNotNull(permit.getId());
        assertTrue(permit.getId().startsWith("PERMIT"));
        assertEquals(car1, permit.getVehicle());
        assertNotNull(permit.getExpirationDate());
        assertNotNull(permit.getRegistrationDate());

        // Verify car was updated with permit info
```

```
✓ 14 tests passed  14 tests total, 32 ms
G:\Java\bin\java.exe ...

Process finished with exit code 0
```

TestTransactionManager.java



```java
package com.parking;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.HashMap;

/**
 * This is the class that manages all the parking transactions.
 */
public class TransactionManager {  4 usages
    private ArrayList<ParkingTransaction> transactions;  5 usages
    private HashMap<String, ArrayList<ParkingTransaction>> vehicleTransaction;  5 usages

    public TransactionManager() {  2 usages
        this.transactions = new ArrayList<>();
        this.vehicleTransaction = new HashMap<>();
    }

    /**
     * This method will create a parking transaction and will add it to the transactions list.
     * @param date The date of the parking event
     * @param permit The parking permit used
     * @param lot The parking lot where the car was parked
     * @return The created ParkingTransaction
     */
    public ParkingTransaction park(Calendar date, ParkingPermit permit, ParkingLot lot) {  29 usages
        Money feeCharged = calculateCharge(permit.getVehicle().getType());
        ParkingTransaction transaction = new ParkingTransaction(date, permit, lot, feeCharged);

        // Add to main transactions list
        transactions.add(transaction);

        // Add to vehicle-specific transaction tracking
        String licensePlate = permit.getVehicle().getLicense();
        if (!vehicleTransaction.containsKey(licensePlate)) {
            vehicleTransaction.put(licensePlate, new ArrayList<>());
        }
```

```java
package com.parking;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import static org.junit.jupiter.api.Assertions.*;
import java.util.ArrayList;
import java.util.Calendar;

public class TestTransactionManager {

    private TransactionManager transactionManager;  44 usages
    private Customer customer;  7 usages
    private Car compactCar;  2 usages
    private Car suvCar;  2 usages
    private ParkingPermit compactPermit;  25 usages
    private ParkingPermit suvPermit;  9 usages
    private ParkingLot lot;  30 usages

    @BeforeEach
    public void setUp() {
        transactionManager = new TransactionManager();

        Address customerAddress = new Address( streetAddress1: "123 Main St",  streetAddress2: "Apt 1",  city: "Boston",  state: "MA",  zipcode: "12345");
        customer = new Customer( customerId: "Customer0",  name: "John Doe", customerAddress,  phoneNumber: "555-1234");

        compactCar = new Car( license: "COMP123", CarType.COMPACT, customer.getCustomerId());
        suvCar = new Car( license: "SUV456", CarType.SUV, customer.getCustomerId());

        Calendar regDate = Calendar.getInstance();
        Calendar expDate = Calendar.getInstance();
        expDate.add(Calendar.YEAR,  amount: 1);

        compactPermit = new ParkingPermit( id: "PERMIT0", compactCar, expDate, regDate);
        suvPermit = new ParkingPermit( id: "PERMIT1", suvCar, expDate, regDate);

        Address lotAddress = new Address( streetAddress1: "100 Park Ave",  streetAddress2: null,  city: "Boston",  state: "MA",  zipcode: "12345");
        lot = new ParkingLot( lotId: "LOT1", lotAddress,  capacity: 100);
    }
```

```
31 ms   ✓ 17 tests passed  17 tests total, 31 ms
31 ms
G:\Java\bin\java.exe ...
Car COMP123 entered lot LOT1
Car COMP123 entered lot LOT2
Car COMP123 entered lot LOT1
Car COMP123 entered lot LOT1
Car SUV456 entered lot LOT1
Car COMP123 entered lot LOT1
Car SUV456 entered lot LOT1
Car COMP123 entered lot LOT1
Car COMP123 entered lot LOT1
Car COMP123 entered lot LOT1
Car COMP123 entered lot LOT1
Car COMP123 entered lot LOT1
Car SUV456 entered lot LOT1
```