

**Using Frameworks for**  
**Master of Science in Information Technology**  
**Software Design and Programming**

James McKenna

University of Denver College of Professional Studies

11/07/2025

Faculty: Nirav Shah, M.S.

Director: Cathie Wilson, M.S.

Dean: Michael J. McGuire, MLS

### **Abstract**

This portion of our class assignment is to continue making use of our Parking System class that will enhance the functionality and design that we've built upon in previous assignments. This time the focus was to use different methods such as equals or hashCode that utilized any List from all our classes. This design decision not only allowed decoupling in our classes it also forced more getters and redeveloping of our classes. This allowed safer practices such as encapsulation, coupling, and flexibility. This allows our code to have more flexibility, enhanced design, and ease of use when adding even further capabilities to our code.

## CONTENTS

1. What did you find difficult or easy?	3
2. What helped you?	3
3. What did you wish you knew before?	4
4. Outline any implementation decisions and the reasoning behind those.	5
5. Include screenshots of the successful code compilation and test execution.	7

**What did you find difficult or easy?**

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.

**What Helped You?**

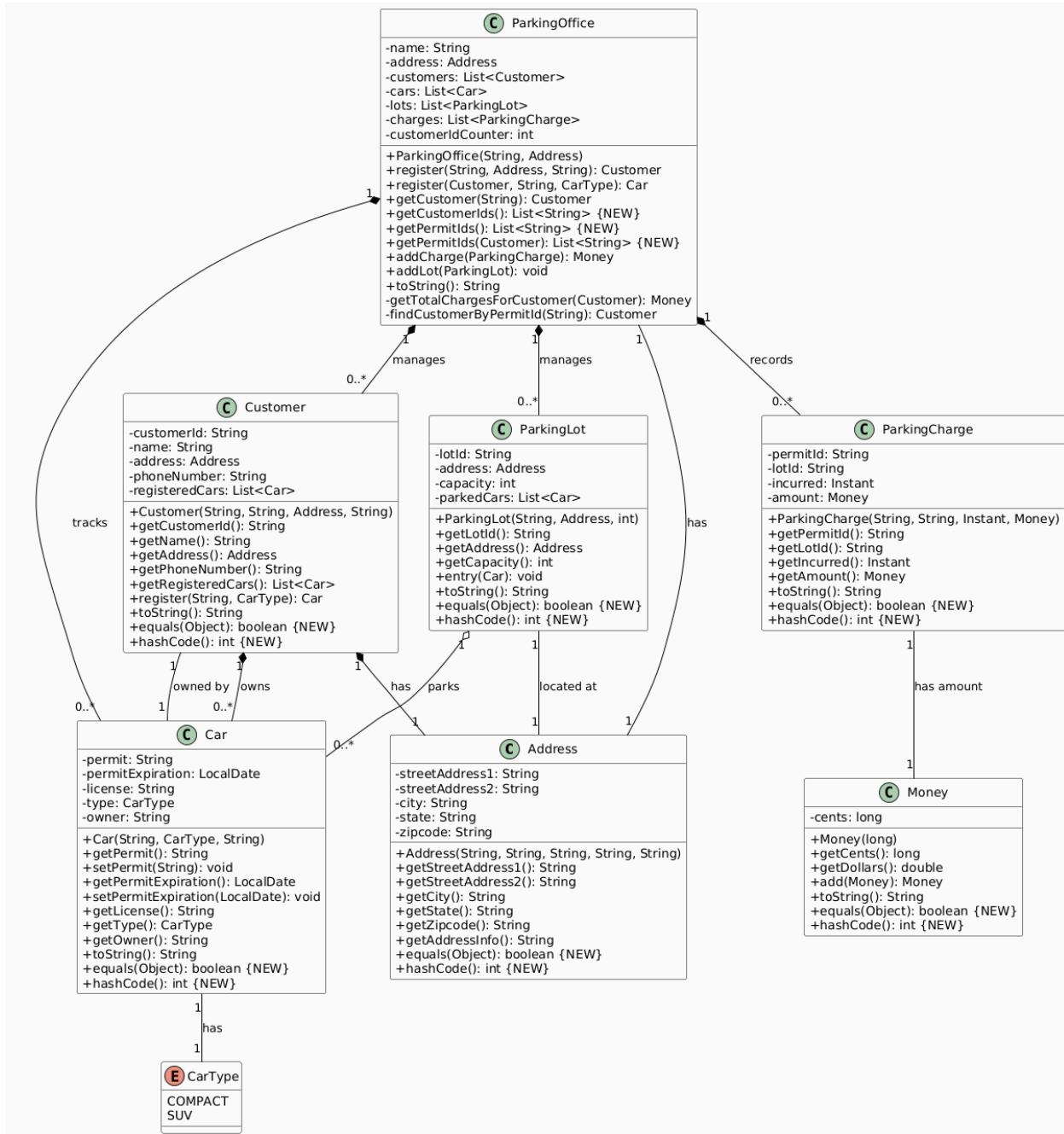
A lot of this assignment was reinforcing previous classes and concepts that we've learned in previous assignments or forcing myself to look it up. I incorporated more encapsulation, more getters, and I required more loose coupling in my code such as the ParkingOffice.java with the getPermitIds method. This method now is only dependent on the getPermitIds, I have tried going through the code with the principle of using the least amount of information as possible from other classes. Also now using the equals method this was a little tricky. I had to look up the importance of using equals in terms of the instantiation for our objects that we're passing in, making sure they're not null, and the we're comparing various fields in between our objects. This took a little time to incorporate as well as test. I also was grateful for previous unit tests as I was able to test further functionality and only really needed

to tweak my unit tests so that we're testing the equals and hashCode methods I added in because of this assignment.

### **What did you wish you knew before?**

Personally I wish I knew more about proper software designs and patterns before this assignment. I had to look a lot of information up specifically about coupling, using equals and hashCode and why this specifically matters. I was aware of dictionaries in Python which are pretty much similar to a Hash Map where we're using this in our hashCode methods. At one point I wasn't quite sure why this was being incorporated but now I understand its in order to reduce look up time and make our code more efficient by explicitly using a hash in order to do look ups. Another key component I was wondering about was why we needed an equals method. I had to look up more information from our class overview lessons and further information online. I realized that we needed an equals method because we needed to prove that the object we were creating and using was the same, and not a different one in a different memory address; even if they were instantiated using the same attributes. Thus the equals method was for saying if I had the same customer with the same ID it's the same customer. My equals method now allows for comparison with customer IDs, it can detect duplicates, can be used with our collections we defined and can remove by value as well.

## Implementations and Design Decisions



I've briefly talked about it before, but overall the major design and architecture didn't really change between all the classes I've created. I primarily added new getters with our ParkingOffice.java such as the getCustomerIds, the getPermitIds, and getPermitIds(Customer). Across all the other classes I primarily added hashCode and equals into all the classes that utilized lists in order to make sure that our classes that are being used in lists can be found and looked up appropriately. I also stopped using a diagram I was making by hand for our UML Class diagram and made my own through PlantUML. It was getting harder and harder to keep all the methods and attributes for all our classes as we kept growing our system for our Parking Office and class.

## Successful Unit Test Screenshots

### TestAddress.java

The screenshot shows the IntelliJ IDEA interface with the TestAddress.java file open. The code implements a Customer class and a TestCustomer class for testing address-related methods. The test cases include registering a customer with a license and car type, getting a customer by ID, and getting all customers. The test results at the bottom show 19 tests passed.

```

public class Customer {
    // Implementation details
}

public class TestCustomer {
    // Implementation details
}

public void testRegisterCustomer() {
    Address address = new Address("123 St.", "Boston", "MA", "12345");
    Customer customer = customer.registerCustomer(address);
    assertEquals("123 St.", customer.getAddress().getStreet());
    assertEquals("Boston", customer.getAddress().getCity());
    assertEquals("MA", customer.getAddress().getState());
    assertEquals("12345", customer.getAddress().getZip());
}

public void testGetCustomerById() {
    Customer customer = customer.registerCustomer();
    Customer customerById = customer.getCustomerById(customer.getId());
    assertEquals(customer, customerById);
}

public void testGetAllCustomers() {
    List<Customer> customers = customer.getAllCustomers();
    assertEquals(1, customers.size());
    assertEquals(customer, customers.get(0));
}
}

```

### TestCar.java

The screenshot shows the IntelliJ IDEA interface with the TestCar.java file open. The code implements a ParkingOffice class and a TestParkingOffice class for testing car-related methods. The test cases include registering a customer with a license and car type, getting a customer by ID, and getting all customers. The test results at the bottom show 19 tests passed.

```

public class ParkingOffice {
    // Implementation details
}

public class TestParkingOffice {
    // Implementation details
}

public void testRegisterCustomer() {
    Car car = new Car("12345", "Compact", "Jesse");
    Customer customer = customer.registerCustomer(car);
    assertEquals("12345", customer.getCar().getLicense());
    assertEquals("Compact", customer.getCar().getType());
    assertEquals("Jesse", customer.getCar().getName());
}

public void testGetCustomerById() {
    Customer customer = customer.registerCustomer();
    Customer customerById = customer.getCustomerById(customer.getId());
    assertEquals(customer, customerById);
}

public void testGetAllCustomers() {
    List<Customer> customers = customer.getAllCustomers();
    assertEquals(1, customers.size());
    assertEquals(customer, customers.get(0));
}
}

```

## TestCustomer.java

```

public class ParkingOffice {
    // ...
}

public class Customer {
    // ...
}

public class TestCustomer {
    // ...
}

```

The screenshot shows the Eclipse IDE interface with the TestCustomer.java file open. The code defines a `ParkingOffice` class with a `registerCustomer` method that takes a `Customer` object and a `CarType`. It also has a `getCustomerById` method that takes an `int` and returns a `Customer`. The `Customer` class is defined elsewhere. The `TestCustomer` class contains several test methods starting with `test`, such as `testRegisterCustomer`, `testGetCustomerById`, etc. The status bar at the bottom indicates "10 tests passed".

## TestMoney.java

```

public class Money {
    // ...
}

public class TestMoney {
    // ...
}

```

The screenshot shows the Eclipse IDE interface with the TestMoney.java file open. The code defines a `Money` class with a constructor taking a double value and a `getDoubleValue` method. The `TestMoney` class contains several test methods starting with `test`, such as `testCreateMoney`, `testGetDoubleValue`, etc. The status bar at the bottom indicates "16 tests passed".

## TestParkingCharge.java

The screenshot shows an IDE interface with the following details:

- Project:** KT4405
- File:** TestParkingCharge.java
- Code Snippet:**

```

public class ParkingOffice {
    ...
    public Car registerCustomer(Customer customer, String license, Cartype type) {
        ...
        return car;
    }

    ...
    public Customer getCustomer(String customerID) {
        ...
        return customer;
    }

    ...
    public List<String> getCustomerIDs() {
        ...
        return customerIDs;
    }
}

```
- Test Class:** TestParkingCharge
- Test Methods:**
  - testParkingChargeConstructor()
  - testParkingChargeRegisterCustomer()
  - testParkingChargeGetCustomer()
  - testParkingChargeGetCustomerIDs()
  - testParkingChargeDeleteCustomer()
  - testParkingChargeUpdateCustomer()
  - testParkingChargeDeleteCustomerID()
- Test Results:**
  - 17 tests passed
  - 0 tests failed
  - 0 tests skipped
- Run Summary:** Process finished with exit code 0

## TestParkingLot.java

The screenshot shows an IDE interface with the following details:

- Project:** KT4405
- File:** TestParkingLot.java
- Code Snippet:**

```

public class ParkingOffice {
    ...
    public void registerCustomer(Customer customer, String license, Cartype type) {
        ...
        return car;
    }

    ...
    public Customer getCustomer(String customerID) {
        ...
        return customer;
    }

    ...
    public List<String> getCustomerIDs() {
        ...
        return customerIDs;
    }
}

```
- Test Class:** TestParkingLot
- Test Methods:**
  - testParkingLotConstructor()
  - testParkingLotRegisterCustomer()
  - testParkingLotGetCustomer()
  - testParkingLotGetCustomerIDs()
  - testParkingLotDeleteCustomer()
  - testParkingLotUpdateCustomer()
  - testParkingLotDeleteCustomerID()
- Test Results:**
  - 18 tests passed
  - 0 tests failed
  - 0 tests skipped
- Run Summary:** Process finished with exit code 0

## TestParkingOffice.java

The screenshot shows an IDE interface with several tabs open. On the left, there's a project tree with packages like Address, Car, Customer, ParkingOffice, and ParkingCharge. The main editor window contains the code for `TestParkingOffice.java`. The code includes methods for registering customers, getting customers by ID, and getting a collection of all customers. It also includes methods for parking cars and calculating charges. The right side of the screen shows the test results for `TestParkingOffice`, indicating 34 tests passed in 14 ms.

```

public class TestParkingOffice {
    // ...
    public void registerCustomer (String name, String license, CarType type) {
        // ...
        car.setCustomer(customer);
        car.setType(type);
        car.setLicense(license);
        car.setParkingOffice(parkingOffice);
        parkingOffice.registerCustomer(name, license, type, car);
        return car;
    }
    // ...
    public Customer getCustomer (String customerId) {
        // ...
        for (Customer customer : customers) {
            if (customer.getId().equals(customerId)) {
                return customer;
            }
        }
        return null;
    }
    // ...
    public List<Customer> getAllCustomer () {
        // ...
        return customers;
    }
    // ...
    public int parkingFee (Customer customer) {
        List<Customer> customers = getAllCustomer();
        for (Customer customer : customers) {
            if (customer.getId().equals(customer.getId())) {
                return customer.getFee();
            }
        }
        return null;
    }
    // ...
}

```

```

public void testRegisterCustomer () {
    // ...
    parkingOffice.registerCustomer("John Doe", "1234567890", "Sedan");
    assertSame("John Doe", parkingOffice.getCustomer("1234567890").getName());
    assertEquals("Sedan", parkingOffice.getCustomer("1234567890").getCarType());
    assertEquals("1234567890", parkingOffice.getCustomer("1234567890").getLicense());
    assertEquals("ParkingOffice", parkingOffice.getCustomer("1234567890").getParkingOffice().getName());
}

public void testGetCustomer () {
    // ...
    parkingOffice.registerCustomer("John Doe", "1234567890", "Sedan");
    parkingOffice.registerCustomer("Jane Doe", "0987654321", "SUV");
    assertEquals("John Doe", parkingOffice.getCustomer("1234567890").getName());
    assertEquals("Jane Doe", parkingOffice.getCustomer("0987654321").getName());
}

public void testGetAllCustomer () {
    // ...
    parkingOffice.registerCustomer("John Doe", "1234567890", "Sedan");
    parkingOffice.registerCustomer("Jane Doe", "0987654321", "SUV");
    assertEquals("John Doe", parkingOffice.getAllCustomer().get(0).getName());
    assertEquals("Jane Doe", parkingOffice.getAllCustomer().get(1).getName());
}

public void testParkingFee () {
    // ...
    parkingOffice.registerCustomer("John Doe", "1234567890", "Sedan");
    parkingOffice.registerCustomer("Jane Doe", "0987654321", "SUV");
    assertEquals(10, parkingOffice.parkingFee(parkingOffice.getCustomer("1234567890")));
    assertEquals(15, parkingOffice.parkingFee(parkingOffice.getCustomer("0987654321")));
}

public void testGetFee () {
    // ...
    parkingOffice.registerCustomer("John Doe", "1234567890", "Sedan");
    parkingOffice.registerCustomer("Jane Doe", "0987654321", "SUV");
    assertEquals(10, parkingOffice.getFee(parkingOffice.getCustomer("1234567890")));
    assertEquals(15, parkingOffice.getFee(parkingOffice.getCustomer("0987654321")));
}

```

Run Results:

- ✓ testRegisterCustomer (14 ms)
- ✓ testGetCustomer (14 ms)
- ✓ testGetAllCustomer (2 ms)
- ✓ testParkingFee (2 ms)
- ✓ testGetFee (2 ms)
- ✓ testGetCustomerWithCustomerId (2 ms)
- ✓ testGetFeeWithCustomerId (2 ms)
- ✓ testGetFeeWithInvalidCustomerId (2 ms)
- ✓ testGetFeeWithMultipleCustomers (2 ms)

Process finished with exit code 0