

**Week 1 Assignment 1: Translating UML into Code for**

ICT 4315-1 Object-Oriented Methods & Paradigms II

Jack Hermanson

University of Denver University College

April 6, 2025

Faculty: Nathan Braun, MS

Director: Cathie Wilson, MS

Dean: Michael J. McGuire, MLS

## Table of Contents

<b><i>Introduction .....</i></b>	<b><i>1</i></b>
<b><i>Design .....</i></b>	<b><i>1</i></b>
<b><i>Explanation of Each Class I Wrote .....</i></b>	<b><i>2</i></b>
<b><i>Unit Tests .....</i></b>	<b><i>2</i></b>
<b><i>Challenges, Lessons, and Conclusions .....</i></b>	<b><i>3</i></b>
<b><i>Conclusion .....</i></b>	<b><i>4</i></b>
<b><i>Appendix: Program Compiles and Tests Pass .....</i></b>	<b><i>5</i></b>
<b><i>References.....</i></b>	<b><i>6</i></b>

One problem that this diagram illustrates is the disconnect between the Transaction Manager and the rest of the application. It is unclear to me which class should have an instance of Transaction Manager as a data member.

### **Explanation of Each Class I Wrote**

It would be impossible to remain under the word limit while adequately explaining each class. Starting at the top-left of the UML diagram, Transaction Manager has a list of Parking Transaction objects and can return them to callers by Customer or Permit. Parking Transaction tracks when someone parked and how much they owe for it. Money represents money using cents to avoid floating imprecision. Parking Service exposes commands that take parameters. Commands implement the Command interface. Each Command provides a method that takes parameters that are passed to other classes and methods. Register Car Command implements Command and registers a new Car with the Parking Office. Register Customer Command also implements Command and registers a new Customer with the Parking Office. Parking Office contains a list of Customers and a list of Parking Lots. Address contains basic address information like street and city. Parking Office, Customer, and Parking Lot all have an Address. Besides Address, Customer and Parking Lot mostly just contain basic information about themselves. Permit Manager contains a list of Permits and a method to register new ones. Permit contains a reference to the car it belongs to. Car has a reference to the Customer it belongs to and a Car Type to track if it's compact or SUV.

### **Unit Tests**

While I started a new solution for ICT 4315, I carried over a little bit of code from ICT 4305. Money is one of the classes I carried over because it is already well-developed and tested. I wrote new unit tests for the Parking Service class, testing that each Command runs properly. I test this by setting up a command with arguments separated by equals signs, so each property can be specified individually. This is most clear in the register car command test,

where the test creates and registers a Car, then confirms that the license plate given to it is what was returned. More unit tests will be necessary to ensure that exceptions are thrown when bad parameters are given, but I'd like to confirm that this is the desired behavior first.

### Challenges, Lessons, and Conclusions

I struggled with this assignment seemingly missing context and well-defined requirements. I followed the provided UML to try to make sure my project was in sync with everyone else's, but that left me with several points of confusion.

One minor example is the get daily rate method on the Parking Lot class. This looks like a getter, and it would make sense to save the daily rate as a data member, but that was not the case on the provided UML diagram, so I'm not sure how you'd make different lots have different rates if all you have is a method that takes a car type. You'd need a base rate and then apply the discount to it based on the car type. I had a similar problem with Parking Office.



Another thing that confused me was that the Parking Office method for registering a Customer returns void, while the overload for registering a Car returns a Permit. If registering a Customer doesn't return a reference to anything, and the list of customers is private, do I need to add some kind of getter that returns a read-only version of the list? That's not on the provided diagram, and I didn't want to over-complicate things.

The commands were very confusing. In this example for registering a car, we have to do some nested constructors because of how a Car has a Customer, and a Customer has an Address. The execute method returns a string, not a reference to the permit you create, so there's no way to access the customer or address. It's also unclear to me what the string should be, so I just returned the license plate here. Additionally, it was not clear what the behavior of the check parameters method should be, so I just have it throw an exception if any parameters are missing. This resulted in a lot of magic strings that made it feel messy and unmaintainable.

```

45  //usage new *
46  @Override
47  public String execute(Properties params) {
48      this.checkParameters(params);
49      CarType carType = params.getProperty("carType").equalsIgnoreCase( anotherString: "compact")
50      ? CarType.COMPACT
51      : params.getProperty("carType").equalsIgnoreCase( anotherString: "suv")
52      ? CarType.SUV
53      : null;
54      ParkingPermit permit = this.office.register(
55          new Car(
56              carType,
57              params.getProperty("licensePlate"),
58              new Customer(
59                  params.getProperty("ownerFirstName"),
60                  params.getProperty("ownerLastName"),
61                  params.getProperty("ownerPhoneNumber"),
62                  new Address(
63                      params.getProperty("ownerStreetAddress1"),
64                      params.getProperty("ownerStreetAddress2"),
65                      params.getProperty("ownerCity"),
66                      params.getProperty("ownerState"),
67                      params.getProperty("ownerZip")
68                  )
69              )
70          );
71      return permit.getCar().getLicensePlate(); // todo - what should this actually return?
72  }

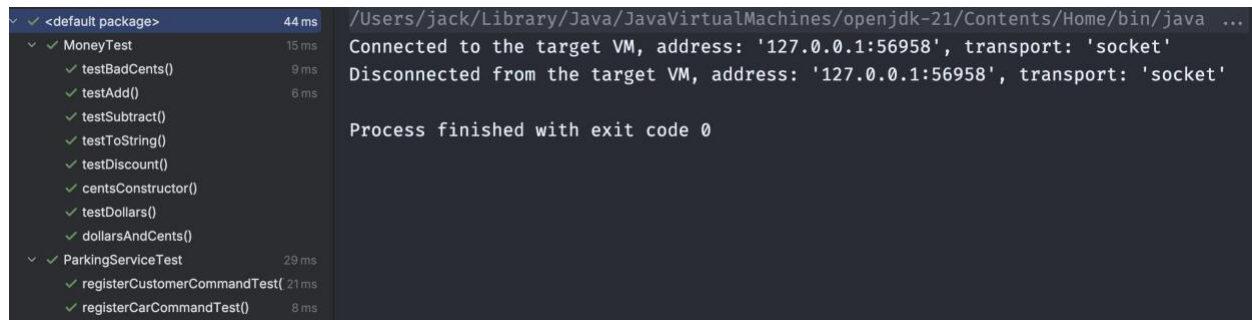
```

## Conclusion

In conclusion, I solved problems by trying different things, looking at the diagram, and reasoning through it as best as I could. However, I am not confident that the provided instructions will always be consistent or reliable, and I expect to experience a lot of frustration going forward due to poorly-defined requirements and assumptions that do not form a coherent idea for how the software is meant to function. I hope that I am wrong about this, and I would be interested in meeting with the instructor to see if we can iron out some of these questions and start the project off on more stable ground. This was not an option in 4305.

## Appendix: Program Compiles and Tests Pass

The program does compile and tests do pass. I am using JDK 21 as indicated in the pom.xml file. I am using the latest version of JUnit but I was very confused when trying to get that to be specifically expressed in the pom.xml file; it just says “RELEASE”.




The screenshot shows an IDE with a test results pane on the left and a terminal window on the right. The test results pane lists the following tests and their durations:

- <default package> 44 ms
- MoneyTest 15 ms
  - testBadCents() 9 ms
  - testAdd() 6 ms
  - testSubtract()
  - testToString()
  - testDiscount()
  - centsConstructor()
  - testDollars()
  - dollarsAndCents()
- ParkingServiceTest 29 ms
  - registerCustomerCommandTest() 21 ms
  - registerCarCommandTest() 8 ms

The terminal window shows the following output:

```
/Users/jack/Library/Java/JavaVirtualMachines/openjdk-21/Contents/Home/bin/java ...  
Connected to the target VM, address: '127.0.0.1:56958', transport: 'socket'  
Disconnected from the target VM, address: '127.0.0.1:56958', transport: 'socket'  
  
Process finished with exit code 0
```



The screenshot shows a snippet of a pom.xml file in an IDE. The code is as follows:

```
<properties>  
  <maven.compiler.source>21</maven.compiler.source>  
  <maven.compiler.target>21</maven.compiler.target>  
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
</properties>  
<dependencies>  
  <dependency>  
    <groupId>org.junit.jupiter</groupId>  
    <artifactId>junit-jupiter</artifactId>  
    <version>RELEASE</version>  
    <scope>test</scope>  
  </dependency>  
</dependencies>
```

## References

JetBrains. 2024. "Getting started with Junit." *JetBrains*. Accessed April 5, 2025.

<https://www.jetbrains.com/help/idea/junit.html>.