**FAST School of Computing**

**Object Oriented Programming – Spring 2025**

**Software Engineering Department**

**LAB 14**

**Polymorphism in C++**

**Learning Outcomes**

In this lab you are expected to learn the following:

* Basic and Adanced Concept and Implementation of Inheritance

**Note:** Plagiarism(from some else or internet) in any 1 question will lead to zero marks in the whole lab task.

**Problem 01:**

In this exercise, you will enhance a simple hourly‐paid employee framework by introducing a new **Commission** type. Imagine you already have a class named Hourly that encapsulates the core details of any hourly worker personal information (name, address, phone number, social security number), a running tally of hours worked, and an hourly pay rate. The Hourly class provides an addHours(double hrs) method to accumulate hours before each payday, a pay() method that multiplies hours worked by the pay rate and then resets the hour count to zero, and a toString() that formats all of this information for display.

Your task is to derive a Commission class from Hourly so that a commissioned employee not only earns for hours worked but also collects a percentage of their sales. To do this, add two new fields: totalSales, a double representing the sum of all sales made in the current period, and commissionRate, another double indicating what fraction of each sales dollar the employee takes home (for example, 0.20 for 20%). Your Commission constructor must accept six arguments: the first five (name, address, phone, SSN, hourly rate) are forwarded to the Hourly constructor, and the sixth initializes commissionRate. You’ll also write a public method

void addSales(double amount) which simply increments totalSales by the given amount.

Next, override the pay() method so that it begins by calling parent class pay(), thereby calculating and zeroing out the hourly wages. Then compute a commission payment as totalSales \* commissionRate, reset totalSales to zero in preparation for the next pay period, and finally return the sum of hourly pay plus commission. Similarly, override display() to first invoke hourly class display()—so you retain all the hourly‐employee details—and then append the current totalSales and commissionRate values, allowing anyone reading the output to see both components of each employee’s earnings.

To verify your implementation, add two Commission objects with fictitious personal data. For instance, one employee might earn $6.25 per hour plus 20% commission, work 35 hours, and report $400 in sales; another might earn $9.75 per hour plus 15% commission, work 40 hours, and bring in $950 in sales. When you run the program, each Commission employee’s console output should display their formatted details, the correct combined pay (hourly wages plus commission), and then demonstrate that both hoursWorked and totalSales have been reset to zero for the next cycle.

**Example output:**

Employee: John Doe

Address: 123 Market St.

Phone: 555-1234

SSN: 111-22-3333

Hourly Rate: $6.25

Hours Worked: 35.0

Total Sales: $400.00

Commission Rate: 0.20

Total Pay this period: $218.75

After pay() call:

Hours Worked: 0.0

Total Sales: $0.00

-------------------------

Employee: Jane Smith

Address: 456 Central Ave.

Phone: 555-5678

SSN: 222-33-4444

Hourly Rate: $9.75

Hours Worked: 40.0

Total Sales: $950.00

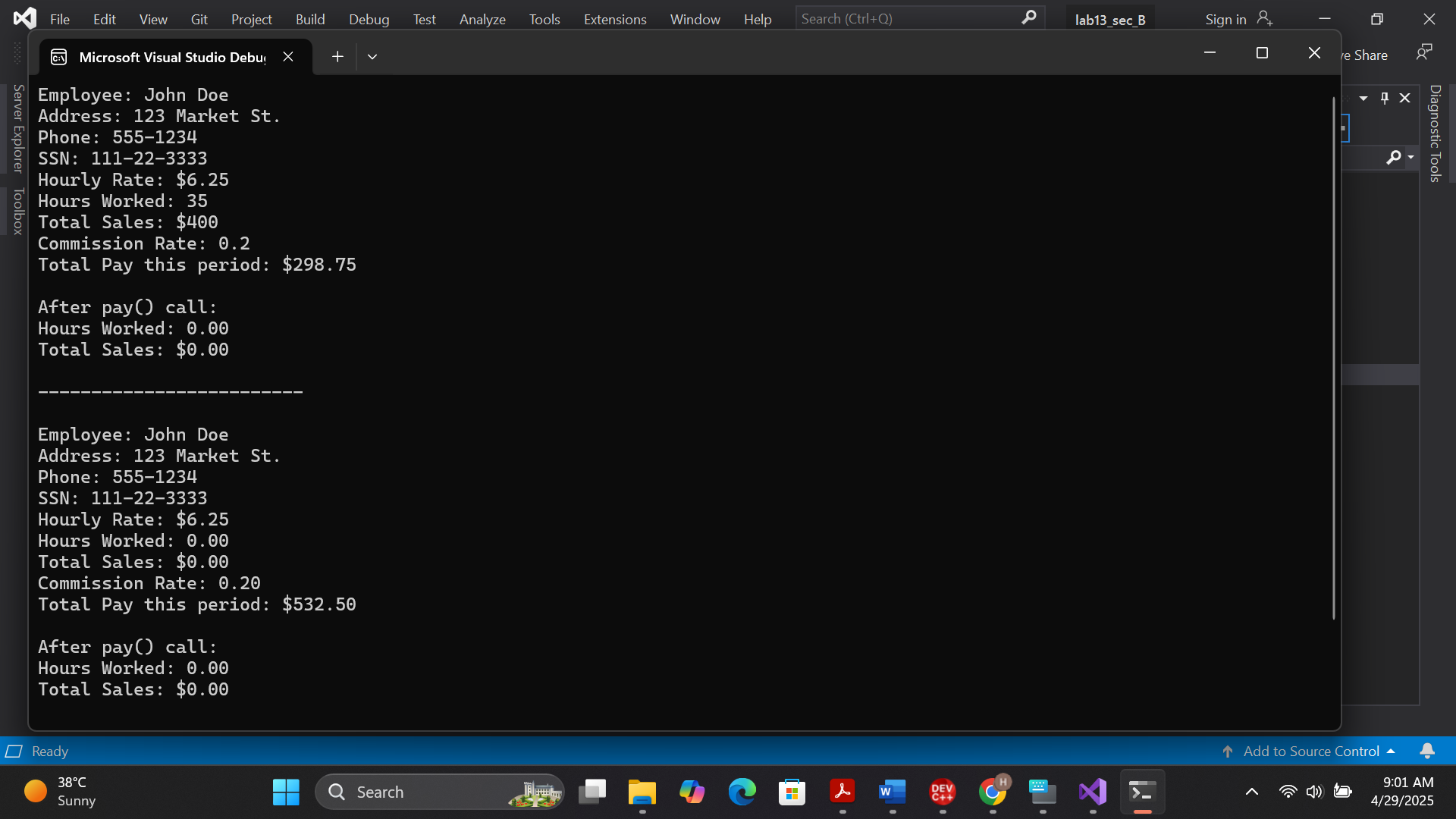
Commission Rate: 0.15

Total Pay this period: $446.25

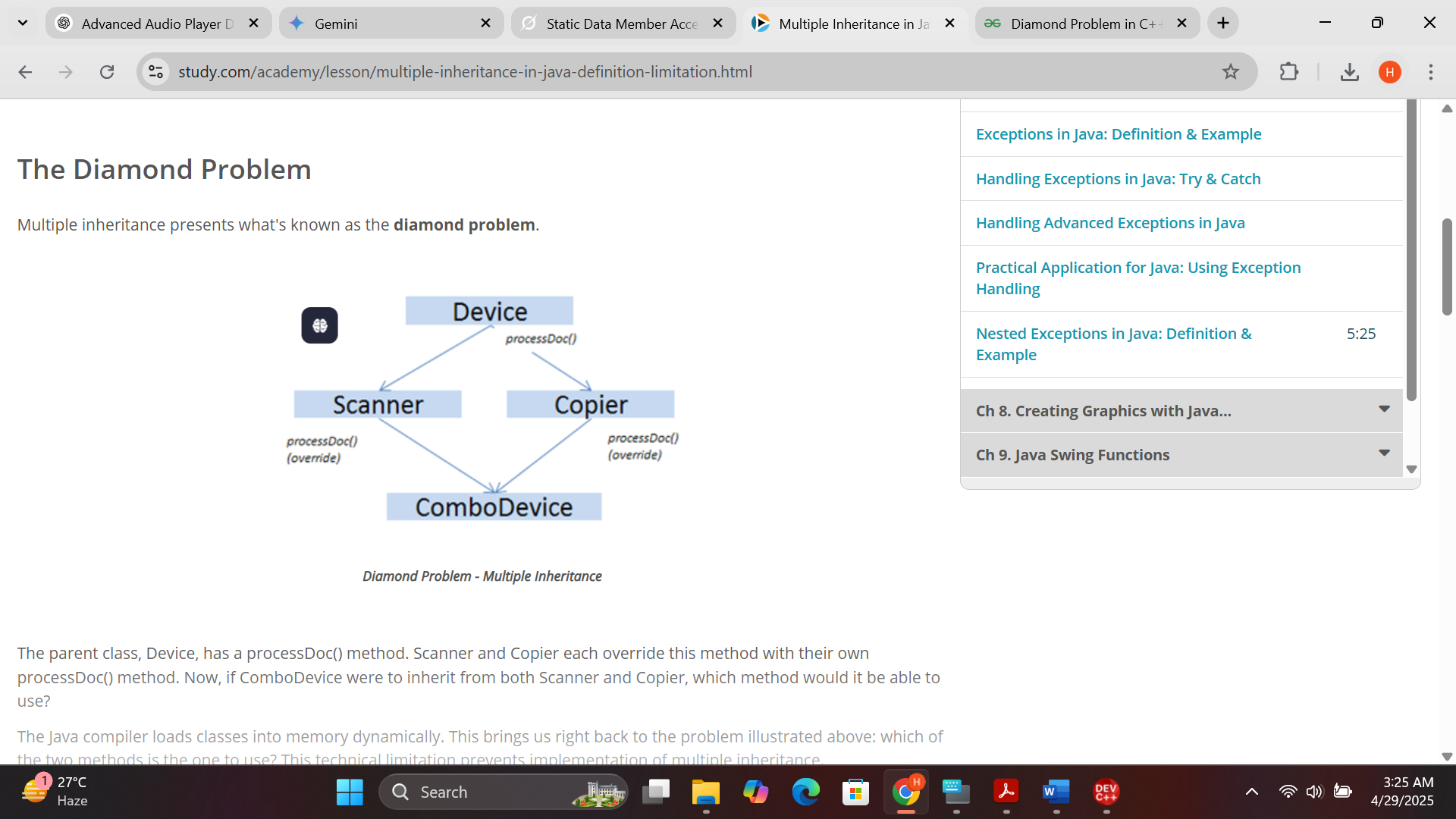
After pay() call:

Hours Worked: 0.0

Total Sales: $0.00



**Problem 02:**



The core issue is: When an object of the ComboDevice class calls the processDoc() method, which version of the method should be executed?

* Does it use the processDoc() method inherited from Scanner?
* Does it use the processDoc() method inherited from Copier?
* Or is there some ambiguity or error because the ComboDevice inherits two different implementations of the same method name from its parent classes? If yes, then how it will be resolved?

**Problem 03:**

(a) **Vehicles**

Write a program that models 2 vehicles (**Car** and **Truck**) and will be able to simulate **driving** and **refueling** them. **Car** and **truck** both have **fuel quantity**, **fuel consumption in liters per km** and can be **driven given distance** and **refueled with given liters.** But in the summer both vehicles use air conditioner and their **fuel consumption** per km is **increased** by **0.9** liters for the **car** and with **1.6** liters for the **truck**. Also the **truck** has a tiny hole in his tank and when it gets **refueled** it gets only **95%** of given **fuel**. The **car** has no problems when refueling and adds **all given fuel to its tank.** If vehicle **cannot** travel given distance its fuel does not change.

**Input**

On the **first line** - information about the car in format **{Car {fuel quantity} {liters per km}}**

On the **second line** – info about the truck in format **{Truck {fuel quantity} {liters per km}}**

On third line - **number of commands N** that will be given on the next **N** lines

On the next **N** lines – commands in format

Drive Car {distance}

D**rive Truck {distance}**

Refuel Car {liters}

Refuel Truck {liters}

**Output**

After each **Drive command** print whether the Car/Truck was able to travel given distance in format if it’s successful. **Print the distance with all digits after the decimal separator except trailing zeros.** Use the **DecimalFormat** class:

**Car/Truck travelled {distance} km**

Or if it is not:

**Car/Truck needs refueling**

Finally print the **remaining fuel** for both car and truck rounded **2 digits after floating point** in format:

**Car: {liters}**

**Truck: {liters}**

|  |  |
| --- | --- |
| **Example Input** | **Output** |
| Car 15 0.3  Truck 100 0.9  4  Drive Car 9  Drive Car 30  Refuel Car 50  Drive Truck 10 | Car travelled 9 km  Car needs refueling  Truck travelled 10 km  Car: 54.20  Truck: 75.00 |
| Car 30.4 0.4  Truck 99.34 0.9  5  Drive Car 500  Drive Car 13.5  Refuel Truck 10.300  Drive Truck 56.2  Refuel Car 100.2 | Car needs refueling  Car travelled 13.5 km  Truck needs refueling  Car: 113.05  Truck: 109.13 |

**Problem 3(b). Vehicles Extension**

Use your solution of the previous task for starting point and add more functionality. Add new vehicle **Bus**. Now every vehicle has **tank capacity** and fuel quantity **cannot fall below 0** (If fuel quantity become less than 0 **print** on the console **“Fuel must be a positive number”**).

The **car** and the **bus cannot be filled** with fuel **more than their tank capacity**. If you **try to put more fuel** in the tank than the **available space,** print on the console **“Cannot fit fuel in tank”** and **do not add any fuel** in vehicles tank.

Add **new command** for the bus. The **bus** can **drive with or without people**. If the bus is driving **with people**, the **air-conditioner is turned on** and its **fuel consumption** per kilometer is **increased with 1.4 liters**. If there are **no people in the bus** when driving the air-conditioner is **turned off** and **does not increase** the fuel consumption.

**Input**

On the first three lines you will receive information about the vehicles in format:

**Vehicle {initial fuel quantity} {liters per km} {tank capacity}**

On fourth line - **number of commands N** that will be given on the next **N** lines

* 1. • On the next **N** lines – commands in format o Drive Car {distance}
  2. o Drive Truck {distance}
  3. o Drive Bus {distance}
  4. o DriveEmpty Bus {distance}
  5. o Refuel Car {liters}
  6. o Refuel Truck {liters}
  7. o Refuel Bus {liters}

**Output** After each **Drive command** print whether the Car/Truck was able to travel given distance in format if it’s successful:

**Car/Truck/Bus travelled {distance} km**

Or if it is not:

**Car/Truck/Bus needs refueling**

If given fuel is **≤ 0** print **“Fuel must be a positive number”.**

If given fuel cannot fit in car or bus tank print **“Cannot fit in tank”**

Finally print the **remaining fuel** for both car and truck rounded **2 digits after floating point** in format:

**Car: {liters}**

**Truck: {liters}**

**Bus: {liters}**

|  |  |
| --- | --- |
| **Example Input** | **Output** |
| Car 30 0.04 70  Truck 100 0.5 300  Bus 40 0.3 150  8  Refuel Car -10  Refuel Truck 0  Refuel Car 10  Refuel Car 300  Drive Bus 10  Refuel Bus 1000  DriveEmpty Bus 100  Refuel Truck 1000 | Fuel must be a positive number  Fuel must be a positive number  Cannot fit fuel in tank  Bus travelled 10 km  Cannot fit fuel in tank  Bus needs refueling  Car: 40.00  Truck: 1050.00  Bus: 37.00 |

**Submission Details:**

1. Save single .cpp file with your roll no and lab number e.g. i22-XXXX\_Lab#.cpp
2. Take screen shot of running test cases of tasks.
3. Zip the .cpp file and screen shots (Do not create .rar file) with roll no and lab no. e.g. i22-XXXX\_Lab#.zip.
4. Submit the zip file on google class room.