Assignment 3: Software Implementation - OO Project with GUI and Data Storage

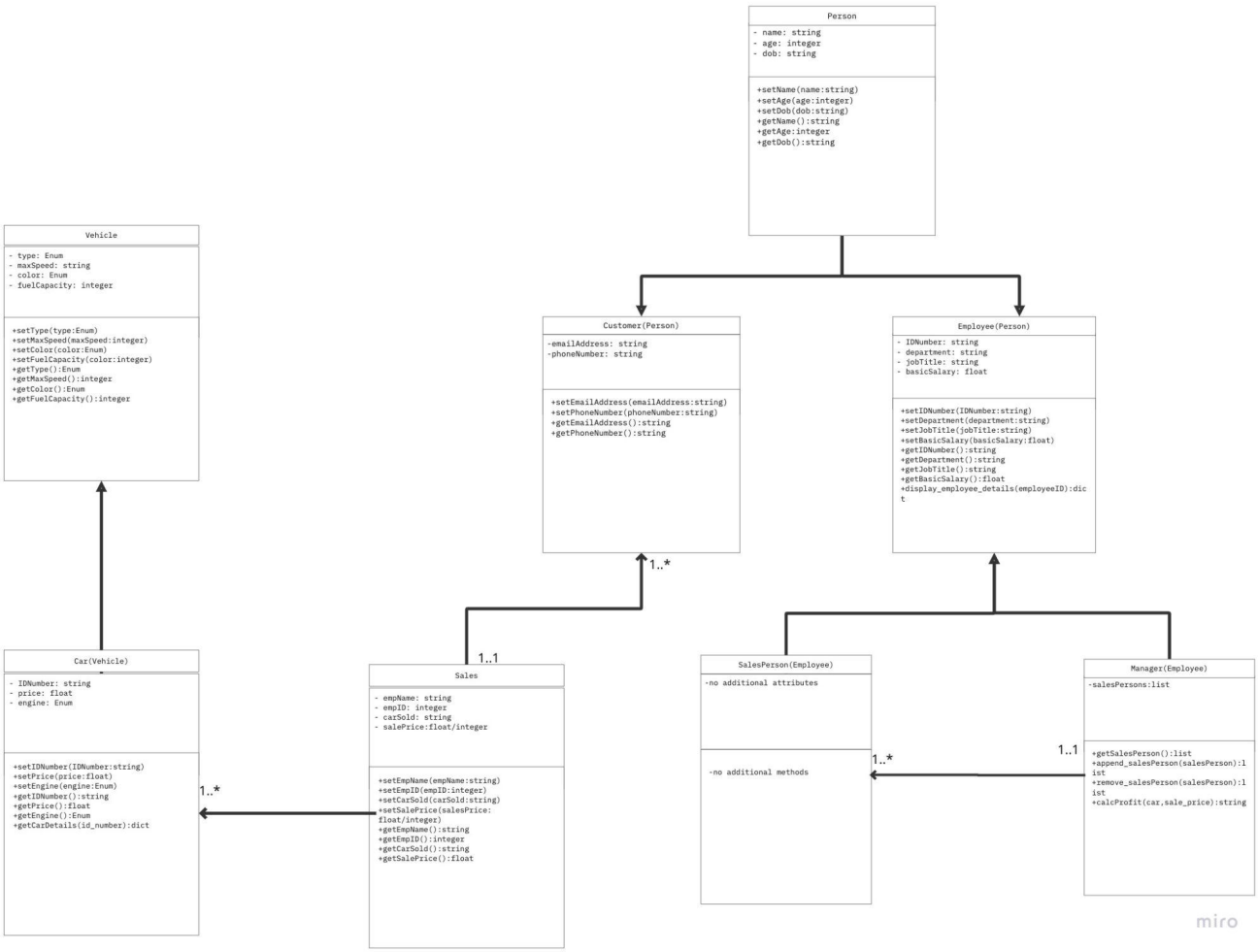
ICS220 - Programming Fundamentals
Professor Kuhail
May 10th, 2023

Moaza Al Falasi
202103349

UML Class Diagram

Link:(in case the image was not clear)

https://miro.com/app/board/uXjVMLpvBpU=/?share_link_id=622441557556



Description

The UML class diagram represents the software for B&M company that manages their employees and cars that they sell. The software system includes several classes that include: Person,Employee, Customer, Manager, Salesperson, Sales, Vehicle, and Car.

Before creating all the classes, I stored the tables that were given as dictionaries for the sales, car details, and employee details as well. These will be used throughout the system to access the data , modify, add, or delete details. I used the pickle module to open them as binary files and write in them and dump them to update the details. Then I loaded the data from pickle files into dictionaries.

First of all, we have the Person class that acts as the parent class for the Employee and Customer class. It has attributes like name , age, date of birth as well as setters and getters. I used basic attributes that belong to a person as I will add more attributes depending on the classes that will inherit from this class.

Next , we have the Customer class that inherits methods and attributes from Person class. It has additional attributes which are emailAddress and phoneNumber as well as setters and getters.
I created a Sales class that has attributes empName, empID, carSold, and salePrice. It has setters and getters and a method that takes in the employee ID and displays all sales of the employee.

Next, we also have a class that inherits from the Person class which the Employee class. It has additional attributes related to an employee which is IDNumber, department, jobTitle , and

basicSalary. I created an Enum class for the department. It has setters and getters for each of the new attributes as well as an additional method that looks for an employee id in the dictionary and prints all the details of that employee given the id number. In case the employee ID is not found, it throws an error.

The Manager class that inherits from the Employee class has the same attributes without additional ones. Since the manager class is in charge of other employees or salesPersons, I create an empty list called salesPersons. I added a getter to get the list of salespersons managed by the manager, and another method that appends or adds salespersons to the list. Another method to remove sales persons from the list was created. A function that takes car and sale price as inputs and returns all profit of manager , salesperson and company. It calculates them based on the percentages that were given.

Next, we have the Salesperson class that inherits from the Employee class all attributes and methods, and it does not have any additional attributes or methods.

Vehicle class acts as the parent class to the Car class and it has attributes like type, maxSpeed, color, and fuelCapacity. It has setters and getters for each attribute.
I created Enum classes for VehicleType, VehicleColor , and EngineType because these are fixed set of values for their respective properties that do not change frequently. It makes it easy to manage the classes and avoid any wrong inputs.

The Car class inherits from the vehicle class its attributes and has additional attributes like IDNumber, price, and engine. I also wrote down the setters, getters, and a method that looks for a car id number in the dictionary and returns all car details if it exists in the dictionary.

Class Relationships:

- **Person and Customer class:** Person class acts as the parent class and customer class inherits attributes and methods of person class. A customer is a person and would have all properties a person would have and additional ones as well
- **Person and Employee class:** Here as well Employee class inherits methods and attributes of Person class. Employee is a person and would have all properties and methods a person has, as well as additional ones.
- **Employee and Manager class:** Employee is the parent class and Manger inherits attributes and methods from Employee class. Manager is an employee
- **Employee and salesperson class:** Salesperson inherits attributes and methods from the Employee class. A salesperson is an employee.
- **Vehicle and Car class:** Vehicle class acts as the parent class in which the Car class inherits methods and attributes as well as additional attributes and a method to display car details. A car is a type of vehicle having all properties of it and additional ones.
- **Manager and SalesPerson class:** They have an association relationship where the manager can have a list of SalesPerson objects that report to them. Manager would access and manage sales data of each salesperson. One manager would be associated with one to many salespersons. One manager for each group of salespeople.
- **Sales and Car class:** Have an association relationship, as the Sales class contains information about the cars that were sold, such as the car's make and model, and the sale price. Sales can have an association with one to many cars.
- **Sales and Customer class:** Could have an association as a sales record contains information about the customer who made the purchase, such as their name, contact information. Each sale would correspond to one customer and each customer may have made one or more purchases.

After creating the classes I created the SoftwareGUI which is responsible for adding, modifying, and deleting details of employees,customers, and cars. I used different widgets such as labels, entries, and buttons. I also used the Top level to create three windows outside the main window, and the message box to either display info or error.

I created the main window that has three buttons in the main page where each button would take the user to either the customer page, employee page , or car page. For each window, there would be entries for the user to input details.

For the customer page, we have entries for name, age, dob, email, and phone number. Then, there are add details, modify details, or delete details buttons which are linked to methods that would do any of these functions. Adding details is done by taking customer information such as name, age, dob, email address, and phone number and creating an object from the Customer class using the given input. The function adds the customer's information to the dictionary with the customer's name as the key and the attributes as the value. Method saves the dictionary to a binary file and displays a success message and clears all boxes. The delete details method looks for the customer's name in the dictionary and if it exists it would delete all customer's details from the dictionary. It then saves the dictionary to the binary file and displays a success message, and clear all boxes. Otherwise, if it is not found, it would display an error. Next, the modifying method would take in the name of the customer , check if it is in the dictionary. If it is, it modifies the customer's attributes using the new values entered as an entry in GUI. Then it saves the dictionary to a binary file, display a success message. In case the customer's name is not found in the dictionary , it would display an error message. I used error handling of inputs that display an error in case invalid input was inputted using the try and except method.

For the employee window, there are entries for name,age, dob, passport, ID number, department, job title, and basic salary. In this window, there are three buttons as well to add, modify , or delete employee details. For the add details, when clicked, it would search for the id

number of the employee in the dictionary, if it is there then it would add the details of the employee and show a message box saying (“Success!”) and clear all entry boxes . Otherwise, if it is not there, it would throw an error. Deleting details button deletes an employee's details from the employee_table dictionary based on the name entered in the employee_name_entry field. If the name is found in the dictionary, it is deleted, and a success message is displayed. If the name is not found, an error message is displayed.It then writes the updated dictionary to a binary file using the pickle module. Finally, it calls the clear_boxes method to clear the input fields. For modifying details, it modifies an existing employee;s information in the dictionary. It retrieves the employee’s data using their name as the key and checks if the employee exists in the dictionary. If the employee exists, the method can update the details based on the user’s input. Then, new details are stored in the dictionary and saved in the file using the pickle module. Success message box is displayed once update is done and boxes are cleared. I used error handling of inputs that display an error in case invalid input was inputted using the try and except method.

For the car window, there are entries for ID number, type, price, maxSpeed, color , fuel capacity, and engine. There are buttons to add, modify , or delete car details. Each of these buttons are connected to commands that would execute the functions. The adding car details method gets the car details from the user interface, creates a new object of Car class with those details, and adds the car to the car_table dictionary that was created before. Then, the updated car_table is written to a file using the pickle module. A message is shown once details are added and boxes are cleared. The delete car details function gets the car type from the user interface and deletes the car from the dictionary if the car type is in the dictionary. Otherwise it would throw an error. Once details are deleted the updated car_table is then written using pickle, a success message is shown, and entry boxes are cleared. Lastly, the modifying car details function gets the car type, retrieves the car object from the car_table dictionary , modifies the car information based

on the user's input, and updates the dictionary with the modified car object. The updated car_table is written to a file using pickle. It shows a success message and clears all entry boxes. In case the car type is not found, it would throw an error.I used error handling of inputs that display an error in case invalid input was inputted using the try and except method.

Python Classes and GUI

```
import tkinter as tk
from tkinter import *
import pickle
from enum import Enum
from tkinter import messagebox
#creates empty dictionary for customer details
cust_dict = {}

#creates dictionary storing all employee details
employee_table = {
   'Susan Meyers':{'Age':29,'DOB': '1994-03-15','Passport': 'ACQ712312','ID
Number': 47899,'Department': 'Accounting','Job Title': 'Manager','Basic
Salary':37500},
   'Mark Jones ':{'Age':25,'DOB': '1998-06-20','Passport': 'BDQ715721', 'ID
Number': 39119,'Department': 'IT','Job Title': 'Salesperson','Basic
Salary':26000},
   'Joy Rogers':{'Age':30,'DOB': '1993-12-01','Passport': 'DEF789012', 'ID
Number': 81774, 'Department': 'Manufacturing', 'Job Title': 'Salesperson',
'Basic Salary': 2400},
}
#creates a dictionary to store all car details
car_table = {
   'Jazz': {'ID Number': 'VX3','Type': 'Hatchback', 'Price': 55000,'Max
Speed': 180, 'Color': 'Red','Fuel Capacity':42},
   'Mark3': {'ID Number': 'SX3','Type': 'Sedan', 'Price': 84000,'Max Speed':
180, 'Color': 'Blue','Fuel Capacity':60},
   'Wagoner' :{'ID Number': 'ZX3','Type': 'SUV', 'Price': 125000,'Max Speed':
220, 'Color': 'Black', 'Fuel Capacity':80}
}
#creates a dictionary to store all sales data
sales_data = [
   {'Employee Name':'Joy Rogers' , 'Employee ID Number': 81774 ,'Car
Sold':'ZX3' , 'Sale Price': 155000},
   {'Employee Name':'Joy Rogers' , 'Employee ID Number': 81774 ,'Car
Sold':'VX3' , 'Sale Price': 57800},
   {'Employee Name': 'Joy Rogers', 'Employee ID Number': 81774, 'Car Sold':
'VX3', 'Sale Price': 55000},
   {'Employee Name': 'Joy Rogers', 'Employee ID Number': 81774, 'Car Sold':
'SX3', 'Sale Price': 89000},
```

```
    {'Employee Name': 'Joy Rogers', 'Employee ID Number': 81774, 'Car Sold':
'SX3', 'Sale Price': 93000},

    {'Employee Name': 'Mark Jones', 'Employee ID Number': 39119, 'Car Sold':
'VX3', 'Sale Price': 58000},
    {'Employee Name': 'Mark Jones', 'Employee ID Number': 39119, 'Car Sold':
'VX3', 'Sale Price': 58000},
    {'Employee Name': 'Mark Jones', 'Employee ID Number': 39119, 'Car Sold':
'VX3', 'Sale Price': 158000},
    {'Employee Name': 'Mark Jones', 'Employee ID Number': 39119, 'Car Sold':
'VX3', 'Sale Price': 158000},
    {'Employee Name': 'Mark Jones', 'Employee ID Number': 39119, 'Car Sold':
'VX3', 'Sale Price': 158000},

]
#load the dictioanry to binary file
with open("employee_table.pkl", "rb") as f:
    employee_table = pickle.load(f)
#load the dictioanry to binary file

with open("car_table.pkl","rb") as f:
    car_table = pickle.load(f)

#creates an enum class for vehicle color
class VehicleColor(Enum):
    Red = 1
    Blue = 2
    Black = 3
    White = 4
    Purple = 5
    Brown = 6
    Silver = 7
    Gold = 8
    Yellow = 9
    Green = 10

#creates an enum class for vehicle type
class VehicleType(Enum):
    Hatchback = 1
    Sedan = 2
    SUV = 3
#creates an enum class for engine type
class EngineType(Enum):
    Diesel = 1
    Petrol = 2
    Electric = 3
    Hybrid = 4
    Gasoline = 5
# creates an enum class for department
```

```
class Department(Enum):
   Accounting = 1
   IT = 2
   Manufacturing = 3
   HR = 4
   Marketing = 5
   Finance = 6


class Person: #Creates a base class called person
   def __init__(self, name,age,dob): #class constructor
       self.name = name #name of the person
       self.age = age #age of the person
       self.dob = dob #date of birth
   #sets the name
   def setName(self,name):
       self.name = name

   # sets the age
   def setAge(self,age):
       self.age = age

   # sets the date of birth
   def setDob(self,dob):
       self.dob = dob

   # get the name
   def getName(self):
       return self.name

   # gets the age
   def getAge(self):
       return self.age
   #gets the date of birth
   def getDob(self):
       return self.dob



#class customer that inherits methods and attributes
class Customer(Person):
   def __init__(self,name,age,dob,emailAddress,phoneNumber):#init intializes
name , age, dob, passportdetails
       super().__init__(name,age, dob)#super used to called method of parent
class
       self.emailAddress = emailAddress
       self.phoneNumber = phoneNumber
   def setEmailAddress(self,emailAddress):
       self.emailAddress = emailAddress
```

```
    def setPhoneNumber(self,phoneNumber):
        self.phoneNumber = phoneNumber
    def getEmailAddress(self):
        return self.emailAddress
    def getPhoneNumber(self):
        return self.phoneNumber


class Employee(Person): #class Employee that inherits attributes & methods
from Person class
    def __init__(self, name,age,dob,
passportDetails,IDNumber,department,jobTitle,basicSalary): #constructor method
taking the following attributes as parameters with new ones as well
        super().__init__(name,age, dob)# constructor of parent class
        self.passportDetails = passportDetails #assigns parameter passed to
constructor attribute of employee class
        self.IDNumber = IDNumber #assigns parameter passed to constructor
attribute of employee class
        self.department = department #assigns parameter passed to constructor
attribute of employee class
        self.jobTitle = jobTitle #assigns parameter passed to constructor
attribute of employee class
        self.basicSalary = basicSalary #assigns parameter passed to constructor
attribute of employee class
    #Setters and getter to set new attributes: IDNumber, department, jobTitle,
and basicSalary
    def setPassportDetails(self,passportDetails):
        self.passportDetails = passportDetails
    def setIDNumber(self,IDNumber):
        self.IDNumber = IDNumber
    def setDepartment(self,department):
        self.department = department
    def setJobTitle(self,jobTitle):
        self.jobTitle = jobTitle
    def setBasicSalary(self,basicSalary):
        self.basicSalary = basicSalary

    def getPassportDetails(self):
        return self.passportDetails
    def getIDNumber(self):
        return self.IDNumber
    def getDepartment(self):
        return self.department
    def getJobTitle(self):
        return self.jobTitle
    def getBasicSalary(self):
        return self.basicSalary
    #given employee id number, it should return all details of employee
    def display_employee_details(self, employeeID):
```

```
        for e in employee_table:
            if employee_table[e]['ID Number'] == employeeID:
                print("Employee Details:")
                return employee_table[e]
    #This function prints all sales given the employee ID
    def salesDetails(self,employeeID):
        totalSales = 0
        salesCount = 0
        for s in sales_data:#loops over data in sales
            if s['Employee ID Number'] == employeeID:#If the given idnumber was
found, it prints all info and increments the salescount
                print(f"Sale Details {salesCount + 1}:")
                print(f"Employee Name: {s['Employee Name']}")
                print(f"Employee ID Number: {s['Employee ID Number']}")
                print(f"Car Sold: {s['Car Sold']}")
                print(f"Sale Price: {s['Sale Price']}")
                salesCount += 1
                totalSales += s['Sale Price']
        if salesCount == 0:#salescount is 0 it prints the following statemment
            print("No sales found for employee")
        else:#otherwise it returns the total sales
            print(f"Total sales: {totalSales}")



class Manager(Employee):#Class Manager that inherits from employee class its
attributes and methods
    def __init__(self,name,age,dob,
passportDetails,IDNumber,department,jobTitle,basicSalary): #constructor method
taking the following attributes as parameters
        super().__init__(name,age, dob,passportDetails, IDNumber, department,
jobTitle, basicSalary ) # constructor of parent class
        self.salesPersons = [] #creates am empty list of salespersons

    #Function that returns list of salespersons managed by manager
    def getSalesPerson(self):
        return self.salesPersons

    #adds a salesperson to the manager's list
    def append_salesPerson(self,salesPerson):
        self.salesPersons.append(salesPerson)

    #removes salesperson from manager's list
    def remove_salesPerson(self,salesPerson):
        self.salesPersons.remove(salesPerson)

    def calcProfit(self,car,sale_price):
        # Method to calculate the profit from selling a car, and distribute it
between the salesperson, manager, and company
```

```
        cost_price = car.getPrice()
        profit = sale_price - cost_price

        salesPersonProfit = profit* 0.065
        managerProfit = profit *0.035
        company_profit = profit - salesPersonProfit - managerProfit
        return f"Sales person's profit: {salesPersonProfit}, Manager's
Profit:{managerProfit}, and Company's Profit:{company_profit}"

class SalesPerson(Employee): #Class SalesPerson that inherits from employee
class its attributes and methods
    def __init__(self,name,age,dob,
passportDetails,IDNumber,department,jobTitle,basicSalary): #constructor method
taking the following attributes as parameters
        super().__init__(name,age, dob,passportDetails, IDNumber, department,
jobTitle, basicSalary ) # constructor of parent class

#Creates class car with attributes , setters, getters, and methods

class Vehicle:
    def __init__(self, type,maxSpeed,color,fuelCapacity):
        self.type = type #type of car
        self.maxSpeed = maxSpeed  # maximum speed of car
        self.color = color  # color of car
        self.fuelCapacity = fuelCapacity  # Fuel capacity of car
    #Setters and getters for all attributes
    def setType(self, type):
        self.type = type
    def setMaxSpeed(self, maxSpeed):
        self.maxSpeed = maxSpeed
    def setColor(self, color):
        self.color = color
    def setFuelCapacity(self, fuelCapacity):
        self.fuelCapacity = fuelCapacity

    def getType(self):
        return self.type
    def getMaxSpeed(self):
        return self.maxSpeed
    def getColor(self):
        return self.color
    def getFuelCapacity(self):
        return self.fuelCapacity
#Creates a class Car that inherits attributes and methods from Vehicle class
class Car(Vehicle):
    def __init__(self,type, maxSpeed, color , fuelCapacity,
IDNumber,price,engine):#constructor method taking the following attributes as
parameters
```

```
        super().__init__(type, maxSpeed, color,fuelCapacity)#use super method,
constructor of parent class
        self.IDNumber = IDNumber #ID number of car
        self.price = price #price of car
        self.engine = engine#engine type of car
    #Setters and getters for new attributes
    def setIDNumber(self,IDNumber):
        self.IDNumber = IDNumber
    def setPrice(self,price):
        self.price = price
    def setEngine(self,engine):
        self.engine = engine

    def getIDNumber(self):
        return self.IDNumber
    def getPrice(self):
        return self.price
    def getEngine(self):
        return self.engine

    #Function that returns all car details given the car id number
    def getCarDetails(self, id_number):
        for car in car_table: #loops over the car table dictionary
            if car_table[car]['ID Number']== id_number:#If the car id number
exists
                print("Car Details:")
                return car_table[car] #it returns all the details of the given
car
        return None#Otherwise it returns None
#Creates a sales class
class Sales:
    def __init__(self,empName,empID, carSold,salePrice):#constructor method
taking the following attributes as parameters
        self.empName = empName #Employee name
        self.empID = empID#Employee id number
        self.carSold = carSold #Types of car sold
        self.salePrice = salePrice#Sales prices
    #Setters and getters for each attribute
    def setEmpName(self,empName):
        self.empName = empName
    def setEmpID(self,empID):
        self.empID = empID
    def setCarSold(self,carSold):
        self.carSold = carSold
    def setSalePrice(self,salePrice):
        self.salePrice = salePrice
    #Function that takes empID as input and dispays all sales given the
employee ID. It would display all sales connected to that employee
    def displayAllSales(self,empID):
```

```
        totalSales = 0 #Intialize to 0
        salesCount = 0#Intialize to 0
        for s in sales_data:  # loops over data in sales
            if s['Employee ID Number'] == empID: # If the given idnumber was
found, it prints all info and increments the salescount and total sales
                    salesCount +=1
                    totalSales += s['Sale Price']
                    print(f"Sale Details {salesCount}:")
                    print(f"Employee Name: {s['Employee Name']}")
                    print(f"Employee ID Number: {s['Employee ID Number']}")
                    print(f"Car Sold: {s['Car Sold']}")
                    print(f"Sale Price: {s['Sale Price']}")
        if salesCount == 0:  # salescount is 0 it prints the following
statemment
            print("No sales found for employee")
        else:  # otherwise it returns the total sales
            print(f"Total sales: {totalSales}")

#creates gui for the Car and Employee management system
class SoftwareGUI:
    def __init__(self):
        #creates a window
        self.root = tk.Tk()
        self.root.geometry("500x300")
        self.root.title("Employee & Car Management System")

        #Label that lets user choose one of the options
        self.option_label = tk.Label(self.root, text = "Choose an option:")
        self.option_label.pack(padx = 10, pady = 10)

        #Creates button to open customer details window
        self.customer_button = tk.Button(self.root,text = "Customers Click
Here",command= self.open_window1)
        self.customer_button.pack(padx = 20, pady = 20)

        #Creates button to open employees details window
        self.staff_button = tk.Button(self.root, text = "Employees Click
Here",command= self.open_window2)
        self.staff_button.pack(padx = 20, pady = 20)

        #Creates button to open car details window
        self.car_button = tk.Button(self.root, text = "Car
Details",command=self.open_window3)
        self.car_button.pack(padx = 20, pady = 20)
    def open_window1(self):
        #Creates a new window for customer details
        window1 = tk.Toplevel(self.root)
        window1.title("Customer Details")
        #creates label and entry for name
```

```
        self.customer_name_label = Label(window1, text="Name:")
        self.customer_name_label.grid(row=0, column=0, padx=5, pady=5)
        self.customer_name_entry = Entry(window1)
        self.customer_name_entry.grid(row=0, column=1, padx=5, pady=5)

        #creates label and entry for Age
        self.customer_age_label = Label(window1, text="Age:")
        self.customer_age_label.grid(row=1, column=0, padx=5, pady=5)
        self.customer_age_entry = Entry(window1)
        self.customer_age_entry.grid(row=1, column=1, padx=5, pady=5)

        # creates label and entry for DOB
        self.customer_dob_label = Label(window1, text="Date of Birth:")
        self.customer_dob_label.grid(row=2, column=0, padx=5, pady=5)
        self.customer_dob_entry = Entry(window1)
        self.customer_dob_entry.grid(row=2, column=1, padx=5, pady=5)

        # creates label and entry for Email Address
        self.customer_emailAddress_label = Label(window1, text="Email
Address:")
        self.customer_emailAddress_label.grid(row=3 , column=0, padx=5, pady=5)
        self.customer_emailAddress_entry = Entry(window1)
        self.customer_emailAddress_entry.grid(row=3, column=1, padx=5, pady=5)

        #creates label and entry for Phone Number
        self.customer_phone_number_label = Label(window1, text="Phone Number:")
        self.customer_phone_number_label.grid(row=4, column=0, padx=5, pady=5)
        self.customer_phone_number_entry = Entry(window1)
        self.customer_phone_number_entry.grid(row=4, column=1, padx=5, pady=5)



        #creates buttons for adding, deleting, modifying with commands set as
well
        self.add_button = Button(window1, text="Add details", command =
self.add_customer_details)
        self.add_button.grid(row=6, column=0, padx=5, pady=5)
        self.delete_button = Button(window1, text="Delete details", command =
self.delete_customer_details)
        self.delete_button.grid(row=6, column=1, padx=5, pady=5)
        self.modify_button = Button(window1, text="Modify details", command =
self.modify_customer_details)
        self.modify_button.grid(row=7, column=0, padx=5, pady=5)
    #Function that clears all entry boxes
    def clear_boxes1(self):
        self.customer_name_entry.delete(0,tk.END)
        self.customer_age_entry.delete(0,tk.END)
        self.customer_dob_entry.delete(0,tk.END)
        self.customer_emailAddress_entry.delete(0,tk.END)
```

```
        self.customer_phone_number_entry.delete(0,tk.END)
    #Function that gets all customer details from the entries
    def add_customer_details(self):
        #get customer details from inputs
        name = self.customer_name_entry.get()
        age = self.customer_age_entry.get()
        dob = self.customer_dob_entry.get()
        email_address = self.customer_emailAddress_entry.get()
        phoneNumber = self.customer_phone_number_entry.get()
        try:
            #check if age is a valid integer between 18 and 100
            age = int(age)
            if age < 18 or age>100:
                raise ValueError("Age must be between 18 and 100")
        except ValueError as e:
            #show an error message box is age input is invalid
            messagebox.showerror("Invalid Input",str(e))
            return
        #check if phone number input is a valid 10 digint number and throws an
error if its not
        if not phoneNumber.isdigit() or len(phoneNumber) != 10:
            messagebox.showerror("Invalid Input","Phone number must be 10
digits")
            return
        #create a new Customer object with input details
        new_customer = Customer(name,age,dob,email_address,phoneNumber)

        #add new customer to the customer dictionary
        cust_dict[new_customer.name] = new_customer.__dict__

        # Write the updated customer dictionary to a binary file using Pickle
        with open("cust_dict.pk1","wb") as f:
            pickle.dump(cust_dict,f)
        # Show a success message box and clear the GUI inputs
        messagebox.showinfo("Success!","Customer details added successfully!")
        self.clear_boxes1()
    def delete_customer_details(self):
        #Gets the customer's name
        cust_name = self.customer_name_entry.get()
        try:
            #checks if the customer name was not inputted and raises an error
if it was not
            if not cust_name:
                raise ValueError("Please enter name of customer to delete.")
        except ValueError as e:
            #Shows error message box of invalid input
            messagebox.showerror("Invalid Input",str(e))
        #if customername is in dictionary , it deletes it's details, shows
message of success
```

```
        if cust_name in cust_dict:
            del cust_dict[cust_name]
            messagebox.showinfo("Success!","Customer details were deleted
successfully!")
        else:
            messagebox.showerror("Error!","Customer details were not found.")
            #Otherwise it shows an error is customer details were not found

        # Write the updated customer dictionary to a binary file using Pickle,
clear the GUI inputs
        with open("cust_dict.pk1","wb") as f:
            pickle.dump(cust_dict,f)
        self.clear_boxes()

    def modify_customer_details(self):
        # Get the name of the customer to be modified from the GUI
        cust_name = self.customer_name_entry.get()
        # Check if the customer exists in the customer dictionary
        cust_data = cust_dict.get(cust_name)
        if cust_data is None:
            messagebox.showerror("Error","Customer was not found.")
        else:
            # If the customer exists, modify their details
            try:
                # Convert the age entered in the GUI to an integer and check if
it is valid
                cust_age = int(self.customer_age_entry.get())
                if cust_age <18 or cust_age >100:
                    # Display an error message if the age entered is not valid
                    raise ValueError("Age must be between 18 adn 100")
            except ValueError as e:
                # Display an error message if the age entered is not valid
                messagebox.showerror("Invalid input",str(e))

            else:
                # If the age entered is valid, update the customer's details in
the dictionary
                cust_data["Age"] = self.customer_age_entry.get()
                cust_data["Date of Birth"] = self.customer_dob_entry.get()
                cust_data["Email Address"] =
self.customer_emailAddress_entry.get()
                cust_data["Phone Number"] =
self.customer_phone_number_entry.get()

                # Update the customer dictionary with the modified customer
details
                cust_dict[cust_name] = cust_data
```

```
                # Save the updated customer dictionary to a binary file using
Pickle
                with open("cust_dict.pk1","wb") as file:
                    pickle.dump(cust_dict,file)
                # Display a success message if the customer details were
updated successfully
                messagebox.showinfo("Success!","Customer details were updated
successfully!")
        # Clear the input boxes in the GUI
        self.clear_boxes()




    def open_window2(self):
        #creates window for employee details
        window2 = tk.Toplevel(self.root)
        window2.title("Employee Details")
        #creates label and entry for name
        self.employee_name_label = Label(window2,text= "Name:")
        self.employee_name_label.grid(row = 0 , column = 0 ,padx =5 , pady =5)
        self.employee_name_entry = Entry(window2)
        self.employee_name_entry.grid(row = 0, column= 1,padx =5 , pady =5)

        # creates label and entry for age
        self.employee_age_label = Label(window2,text="Age:")
        self.employee_age_label.grid(row = 1 , column = 0 ,padx =5 , pady =5)
        self.employee_age_entry = Entry(window2)
        self.employee_age_entry.grid(row = 1, column= 1,padx =5 , pady =5)

        # creates label and entry for DOB
        self.employee_dob_label = Label(window2, text="Date of Birth:")
        self.employee_dob_label.grid(row = 2 , column = 0,padx =5 , pady =5 )
        self.employee_dob_entry = Entry(window2)
        self.employee_dob_entry.grid(row = 2, column= 1,padx =5 , pady =5)

        # creates label and entry for passport details
        self.employee_passport_details_label = Label(window2, text="Passport
Details:")
        self.employee_passport_details_label.grid(row = 3 , column = 0,padx =5
, pady =5 )
        self.employee_passport_details_entry = Entry(window2)
        self.employee_passport_details_entry.grid(row = 3, column= 1,padx =5 ,
pady =5)

        # creates label and entry for ID Number
        self.employee_IDNumber_employee_label = Label(window2, text="ID
Number:")
```

```
        self.employee_IDNumber_employee_label.grid(row = 4 , column = 0,padx =5
, pady =5 )
        self.employee_IDNumber_employee_entry = Entry(window2)
        self.employee_IDNumber_employee_entry.grid(row = 4, column= 1,padx =5 ,
pady =5)

        # creates label and entry for department
        self.employee_department_label = Label(window2,text = "Department: ")
        self.employee_department_label.grid(row = 5 , column = 0 ,padx =5 ,
pady =5)
        self.employee_department_entry = Entry(window2)
        self.employee_department_entry.grid(row = 5, column= 1,padx =5 , pady
=5)

        # creates label and entry for job title
        self.employee_jobTitle_label = Label(window2, text = "Job Title:")
        self.employee_jobTitle_label.grid(row = 6, column = 0,padx =5 , pady =5
)
        self.employee_jobTitle_entry = Entry(window2)
        self.employee_jobTitle_entry.grid(row = 6, column= 1,padx =5 , pady =5)

        # creates label and entry for basic salary
        self.employee_basicSalary_label = Label(window2,text = "Basic Salary:
")
        self.employee_basicSalary_label.grid(row = 7 , column = 0,padx =5 ,
pady =5 )
        self.employee_basicSalary_entry = Entry(window2)
        self.employee_basicSalary_entry.grid(row = 7, column= 1,padx =5 , pady
=5)

        #Creates buttons for adding details, deleting details, modifying detail
with commands as well
        self.add_button = Button(window2,text= "Add details", command =
self.add_employee_details)
        self.add_button.grid(row = 8, column = 0,padx =5 , pady =5)
        self.delete_button = Button(window2, text = "Delete details", command =
self.delete_employee_details)
        self.delete_button.grid(row = 8, column = 1,padx =5 , pady =5)
        self.modify_button = Button(window2, text = "Modify details", command =
self.modify_employee_details)
        self.modify_button.grid(row = 9, column = 0,padx =5 , pady =5)
    #Function that clear all entries
    def clear_boxes(self):
        self.employee_name_entry.delete(0,tk.END)
        self.employee_age_entry.delete(0,tk.END)
        self.employee_dob_entry.delete(0,tk.END)
        self.employee_passport_details_entry.delete(0,tk.END)
        self.employee_IDNumber_employee_entry.delete(0,tk.END)
        self.employee_department_entry.delete(0,tk.END)
```

```
        self.employee_jobTitle_entry.delete(0,tk.END)
        self.employee_basicSalary_entry.delete(0,tk.END)
    #function to add employee details
    def add_employee_details(self):
        #Gets all employee details
        name = self.employee_name_entry.get()
        age = self.employee_age_entry.get()
        dob = self.employee_dob_entry.get()
        passport = self.employee_passport_details_entry.get()
        Id_num = self.employee_IDNumber_employee_entry.get()
        dept = self.employee_department_entry.get()
        jobTitle = self.employee_jobTitle_entry.get()
        basicSalary = self.employee_basicSalary_entry.get()
        #Error handling for wrong input for basic salary or age
        try:
            if int(age) < 18 or int(age) > 100: #if age is greater than 100 or
less than 18 it would throw an error
                raise ValueError("Age must be between 18 and 100")
            if int(basicSalary) < 0 :#If the salary is less than 0/negative it
would throw an error
                raise ValueError("Basic salary cannot be negative")
            if len(passport) != 9:#If length of passport not equal to 9 , it
throws an error
                raise ValueError("Passport must be 8 characters")
            if len(Id_num) != 5:#If length of ID number not equal to 5 , it
throws an error
                raise ValueError("Id number must 5 digits")
            if not
Department.__members__.get(self.employee_department_entry.get()):#IF the
department is not found in the Enum class it throws an error
                raise ValueError("Department cannot be found")
            new_emp = Employee(name, age, dob, passport, Id_num, dept,
jobTitle, basicSalary)#creates new Employee object using the details
            employee_table[new_emp.name] = new_emp.__dict__ #adds it  to the
dictionary
            with open("employee_table.pk1", "wb") as f:# opens the binary file
in write mode
                pickle.dump(employee_table, f)# saves the updated
employee_table to the file using pickle
            #Shows message box of success and clears all boxes
            messagebox.showinfo("Success!", "Employee details added
successfully!")
            self.clear_boxes()
        #display error message  in case of invalid input
        except ValueError as e:
            messagebox.showerror("Invalid Input", str(e)) #messagebox appears
in case of wrong inpuy
```

```
   def delete_employee_details(self):
       employee_name = self.employee_name_entry.get()
       try:
           if not employee_name:
               raise ValueError("Please enter the name of the employee to
delete.")
       except ValueError as e:
           messagebox.showerror("Invalid Input",str(e))

       if employee_name in employee_table:
           del employee_table[employee_name]
           messagebox.showinfo("Success!", "Employee details were deleted
successfully!")
       else:
           messagebox.showerror("Error!", "Employee was not found")
       with open("employee_table.pk1", "wb") as file:
           pickle.dump(employee_table, file)
       self.clear_boxes()
   #Deleted en employee's details from the dictioanry
   def modify_employee_details(self):
       # gets the name of the employee to be deleted from the user input
       emp_name = self.employee_name_entry.get()
       emp_data = employee_table.get(emp_name)
       if emp_data is None:
           # raises an error if the employee name is not provided
           messagebox.showerror("Error","Employee was not found")
       else:
           try:
               #Gets the entries for employee age, passport, id number, and
basic salary
               emp_age = int(self.employee_age_entry.get())
               emp_pass = self.employee_passport_details_entry.get()
               emp_id_number = self.employee_IDNumber_employee_entry.get()
               emp_basicSalary = float(self.employee_basicSalary_entry.get())
               #Throws an error if employee age is less than 18 or greater
than 100
               if emp_age < 18 or emp_age >100:
                   raise ValueError("Age must be between 18 and 100")
               #Throws an error if length of passport is not equal to 9
               if len(emp_pass) != 9:
                   raise ValueError("Passport must be 8 characters")
               # Throws an error if length id number is not equal to 5
               if len(emp_id_number) != 5:
                   raise ValueError("Id number must 5 digits")
               # Throws an error is basic salary is negative
               if emp_basicSalary <0:
                   raise ValueError("Basic salary cannot be negative")
               #Throws an error is inputted dept is not found in  enum class
```

```
                if not
Department.__members__.get(self.employee_department_entry.get()):
                    raise ValueError("Department cannot be found")
            except ValueError as e:
                # displays error message if age is not within a valid range
                messagebox.showerror("Invalid Input",str(e))

            #updates employee data with new values entered in GUI
            emp_data["Age"] = self.employee_age_entry.get()
            emp_data["Date of Birth"] = self.employee_dob_entry.get()
            emp_data["Passport"] = self.employee_passport_details_entry.get()
            emp_data["IDNumber"] = self.employee_IDNumber_employee_entry.get()
            emp_data["Department"] = self.employee_department_entry.get()
            emp_data["Job Title"] = self.employee_jobTitle_entry.get()
            emp_data["Basic Salary"] = self.employee_basicSalary_entry.get()

            employee_table[emp_name] = emp_data#updates employee_table with
modified data
            with open("employee_table.pk1","wb") as f:#updates table in pickle
file
                pickle.dump(car_table,f)
        #Shows messagebox of success and clear all entries
            messagebox.showinfo("Success!","Employee details updated
successfully!")
        self.clear_boxes()

    def open_window3(self):
        #Creates window for car details
        window3 = tk.Toplevel(self.root)
        window3.title("Car Details")

        #creates label and entry for ID Number
        self.car_IDNumber_label = Label(window3, text="ID Number:")
        self.car_IDNumber_label.grid(row=0, column=0, padx=5, pady=5)
        self.car_IDNumber_entry = Entry(window3)
        self.car_IDNumber_entry.grid(row=0, column=1, padx=5, pady=5)

        # creates label and entry for car type
        self.car_type_label = Label(window3, text="Type:")
        self.car_type_label.grid(row=1, column=0, padx=5, pady=5)
        self.car_type_entry = Entry(window3)
        self.car_type_entry.grid(row=1, column=1, padx=5, pady=5)

        # creates label and entry for car price
        self.car_price_label = Label(window3, text="Price:")
        self.car_price_label.grid(row=2, column=0, padx=5, pady=5)
        self.car_price_entry = Entry(window3)
        self.car_price_entry.grid(row=2, column=1, padx=5, pady=5)
```

```
        # creates label and entry for car max speed
        self.car_maxSpeed_label = Label(window3, text="Max Speed:")
        self.car_maxSpeed_label.grid(row=3, column=0, padx=5, pady=5)
        self.car_maxSpeed_entry = Entry(window3)
        self.car_maxSpeed_entry.grid(row=3, column=1, padx=5, pady=5)

        # creates label and entry for car color
        self.car_color_label = Label(window3, text="Color:")
        self.car_color_label.grid(row=4, column=0, padx=5, pady=5)
        self.car_color_entry = Entry(window3)
        self.car_color_entry.grid(row=4, column=1, padx=5, pady=5)

        # creates label and entry for car fuel capacity
        self.car_fuelCapacity_label = Label(window3, text="Fuel Capacity:")
        self.car_fuelCapacity_label.grid(row=5, column=0, padx=5, pady=5)
        self.car_fuelCapacity_entry = Entry(window3)
        self.car_fuelCapacity_entry.grid(row=5, column=1, padx=5, pady=5)

        self.car_engine_label = Label(window3, text="Engine:")
        self.car_engine_label.grid(row=6, column=0, padx=5, pady=5)
        self.car_engine_entry = Entry(window3)
        self.car_engine_entry.grid(row=6, column=1, padx=5, pady=5)

        #Creates buttons for adding details, deleting details, modifying
detail,and commands as well
        self.add_button = Button(window3, text="Add details",
command=self.add_car_details )
        self.add_button.grid(row=7, column=0, padx=5, pady=5)
        self.delete_button = Button(window3, text="Delete details", command =
self.delete_car_details)
        self.delete_button.grid(row=7, column=1, padx=5, pady=5)
        self.modify_button = Button(window3, text="Modify details", command =
self.modify_car_details)
        self.modify_button.grid(row=8, column=0, padx=5, pady=5)
    #Function to clear all boxes once buttons are clicked
    def clear_boxes_car(self):
        self.car_IDNumber_entry.delete(0,tk.END)
        self.car_type_entry.delete(0,tk.END)
        self.car_price_entry.delete(0,tk.END)
        self.car_maxSpeed_entry.delete(0,tk.END)
        self.car_color_entry.delete(0,tk.END)
        self.car_fuelCapacity_entry.delete(0,tk.END)
        self.car_engine_entry.delete(0,tk.END)

    def add_car_details(self):
        #Gets all car details from entries
        carType = self.car_type_entry.get()
        maxSpeed = self.car_maxSpeed_entry.get()
        color = self.car_color_entry.get()
```

```
        fuelCapacity = self.car_fuelCapacity_entry.get()
        idNumber = self.car_IDNumber_entry.get()
        price = self.car_price_entry.get()
        engine = self.car_engine_entry.get()

        try:
            #Checks if the vehicle type can be found in the enum class
            if not VehicleType.__members__.get(self.car_type_entry.get()):
                raise ValueError("Car type cannot be found")

            # Checks if the price is greater than 0
            if int(maxSpeed) >200:
                raise ValueError("Maximum speed cannot be greater than 200")

            #Checks if the vehicle color can be found in the enum class
            if not VehicleColor.__members__.get(self.car_color_entry.get()):
                raise ValueError("Car color cannot be found")
            # Checks if the price is less than 0
            if float(price) < 0:
                raise ValueError("Price cannot be negative")
            if not EngineType.__members__.get(self.car_engine_entry.get()):
                raise ValueError("Engine type was not found")

            # Create a new car object and add to car_table
            new_car =
Car(carType,maxSpeed,color,fuelCapacity,idNumber,price,engine)
            car_table[new_car.type] = new_car.__dict__

            # Write car_table to file
            with open("car_table.pkl","wb") as f:
                pickle.dump(car_table,f)
            #Show success message
            messagebox.showinfo("Success!","Car details added successfully!")
        # Show error message and clear entry boxes
        except ValueError as e:
            messagebox.showerror("Error!",str(e))
        self.clear_boxes_car()

    def delete_car_details(self):
        #get car type to delete
        carType = self.car_type_entry.get()

        # Check if car type is valid
        if not VehicleType.__members__.get(carType):
            raise ValueError("Car type cannot be found")

        # Check if car type is in car_table and delete if it is
        if carType in car_table:
            del car_table[carType]
```

```
            #Shows success message
            messagebox.showinfo("Success!","Car details were deleted
successfully!")
        else:
            #Otherwise throws an error is car type was not found
            messagebox.showerror("Error!","Car type was not found")
        #Write updates car_table to file and clear entry boxes
        with open("car_table.pkl","wb") as file:
            pickle.dump(car_table,file)
        self.clear_boxes_car()

    def modify_car_details(self):
        # Get car type to modify
        car_type = self.car_type_entry.get()
        # Get existing car data
        car_data = car_table.get(car_type)

        # Check if car exists
        if car_data is None:
            messagebox.showerror("Error","Car was not found")
        else:
            try:
                # Validation checks for vehicle type, color, max speed, price
and engine type
                if not VehicleType.__members__.get(car_type):
                    raise ValueError("Car type cannot be found")
                if int(self.car_maxSpeed_entry.get()) > 200:
                    raise ValueError("Maximum speed cannot be greater than
200")
                if not
VehicleColor.__members__.get(self.car_color_entry.get()):
                    raise ValueError("Car color cannot be found")
                if float(self.car_price_entry.get()) < 0:
                    raise ValueError("Price cannot be negative")
                if not EngineType.__members__.get(self.car_engine_entry.get()):
                    raise ValueError("Engine type was not found")

                # Update car data and write to car_table
                car_data["maxSpeed"] = self.car_maxSpeed_entry.get()
                car_data["color"] = self.car_color_entry.get()
                car_data["fuelCapacity"] = self.car_fuelCapacity_entry.get()
                car_data["idNumber"] = self.car_IDNumber_entry.get()
                car_data["price"] = self.car_price_entry.get()
                car_data["engine"] = self.car_engine_entry.get()

                #Updates the values entered in GUI to the dictionary
                car_table[car_type] = car_data

                #Writes dictioanry to the file using pickle
```

```
                with open("car_table.pk1","wb") as f:
                    pickle.dump(car_table,f)
                #Sucess messgae is shown
                messagebox.showinfo("Success!", "Car details updated
successfully!")
            except ValueError as e:
                #Error messgae is shown if any exception is rasied during
validation
                messagebox.showerror("Error", str(e))

#creates an instance of the GUI and start the main loop for the root window
gui = SoftwareGUI()
gui.root.mainloop()

#Tests for car details, with idnumber it returned all car details
car1 =
Car('VX3',VehicleType.Hatchback,55000,180,VehicleColor.Red,42,EngineType.Petro
l)
car_d = car1.getCarDetails('VX3')
print(car_d)
# it displayed all car details given ID number

#Tests for employee details, with idnumber it returned all employee details
empl1 = Employee('Susan
Myers',29,'1994-03-15','ACQ712312',47899,Department.Accounting
,'Manager',37500)
emp_D = empl1.display_employee_details(47899)
print(emp_D)
#It displayed all employee details given the ID number

#Test for displaying sales detail of an employee given ID Number
#For a wrong employee ID it would display no sales found for employee
sl = Sales('Joy Rogers',81773,'ZX3',58000)
sl1 = sl.displayAllSales(81773)
print(sl1)

#IT displayed all sales details given employee ID and total sales as well
sl2 = Sales('Joy Rogers',81774,'ZX3',58000)
sll = sl2.displayAllSales(81774)
print(sll)
sale_price = 60000
manager = Manager("Susan
Myers",29,'1994-03-15','ACQ712312',47899,Department.Accounting,'Manager',37500
)
m = manager.calcProfit(car1,sale_price)
print(m)

#It access the dictionary and prints out the profit for sales for each
manager, salesperson, and company
```

```
car_sold = sales_data[0]['Car Sold']
car = Car(car_sold, 200, 'red', 50, '12345', 10000, 'V8')
for sale in sales_data:
   print(manager.calcProfit(car, sale['Sale Price']))
```

Summary of learnings

I gained a deeper understanding of the concept of object-oriented programming and how it applies to Python classes. I was able to create a UML class diagram to model the different classes and their relationships, including inheritance and association. I learned how to create methods for adding and updating car details and calculating profits for sales made by salespersons.

I also learned how to use external libraries like pickle to save and load data from files, and how to use tkinter to create a graphical user interface for the car company's management system. Additionally, I learned how to handle errors and display user-friendly messages using messagebox. I learned how to use different widgets to create the user interface for the GUI as well as different windows to represent the employees page, customers page, and cars page.
I learned how to connect the gui to the classes that I created to create an object.
I also was able to use commands in the tkinter button to allow the system to either add, delete, or modify details.

Overall, these concepts helped me to better understand Python classes and how to use them to create practical applications, such as a management system for a car company.