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

Alia Lootah

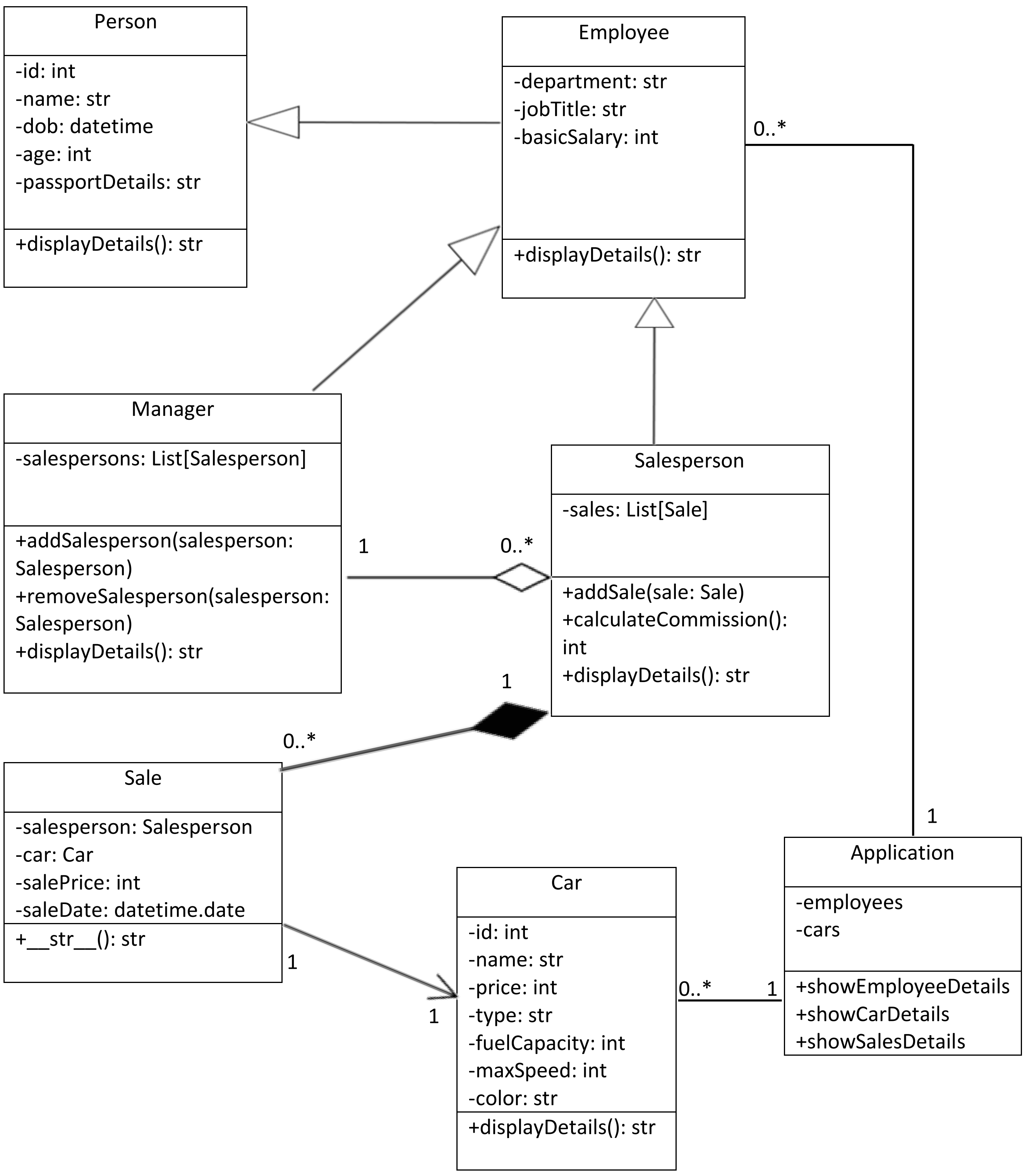
College of Interdisciplinary Studies, Zayed University

ICS220 - 22873 Programming Fundamentals

Professor Kuhail

May 10, 2023

UML class diagram:



Description:

- Application has a 0-to-many relationship with Employee, as an application instance can manage multiple employees (which includes Managers and Salespersons).
- Application has a 0-to-many relationship with Car, as an application instance can manage multiple cars.
- Manager has a 0-to-many aggregation relationship with Salesperson, as one manager can supervise multiple salespersons, but a salesperson might not have a manager assigned.
- Salesperson has a 0-to-many composition relationship with Sale, as one salesperson can have multiple sales, but it's also possible that a salesperson hasn't made any sales.
- Sale has a 1-to-1 relationship with Car, as each sale is linked to a single car.

Inheritance relationships:

- Employee inherits from the Person class.
- Manager and Salesperson inherit from the Employee class. This indicates that these classes share common attributes and behaviors with the Employee class.

Aggregation relationship:

- The Manager class has an aggregation relationship with the Salesperson class. This means that a Manager object can have multiple Salesperson objects associated with it. If a Manager object is destroyed, the associated Salesperson objects are not destroyed but can exist independently.

Composition relationship:

- There is a composition relationship between Salesperson and Sale class. The Salesperson class has a list of Sale objects called sales. This list represents the sales made by the salesperson. The salesperson object owns these sales, and when the salesperson object is destroyed, the sales associated with that salesperson will also be destroyed.

Association relationships:

- The Sale class is associated with Car class. This means that a Sale object is connected to a Car object (which was sold).
- The Application class is associated with both the Employee and Car classes. This means that the Application object interacts with multiple instances of Employee and Car objects.

Assumptions:

- An Employee object can be a Manager or a Salesperson.
- A Manager can have multiple Salesperson objects associated with them.
- A Salesperson can have multiple Sale objects associated with them.
- Each Sale object is associated with exactly one Salesperson and one Car.
- The Application object interacts with multiple Employee and Car objects but does not own them.

Python:

```python
import tkinter as tk
import pickle
import datetime
from typing import List
from enum import Enum


class CarType(Enum):
    Hatch = 1
    Sedan = 2
    SUV = 3


# Person class definition
class Person:
    # Constructor method for the Person class
    def __init__(self, id: int, name: str, dob: datetime.date, age: int,
passportDetails: str):
        self.id = id
        self.name = name
        self.dob = dob
        self.age = age
        self.passportDetails = passportDetails

    # Method to display the details of a Person
    def displayDetails(self) -> str:
        return f"ID: {self.id}, Name: {self.name}, DOB: {self.dob}, Age:
{self.age}, Passport Details: {self.passportDetails}"


# Employee class definition (inherits from Person)
class Employee(Person):
    # Constructor method for the Employee class
    def __init__(self, id: int, name: str, dob: datetime.date, age: int,
passportDetails: str, department: str, jobTitle: str, basicSalary: int):
        super().__init__(id, name, dob, age, passportDetails)
        self.department = department
        self.jobTitle = jobTitle
        self.basicSalary = basicSalary

    # Method to display the details of an Employee (overrides Person's
displayDetails)
    def displayDetails(self) -> str:
        return f"{super().displayDetails()}, Department: {self.department},
Job Title: {self.jobTitle}, Basic Salary: {self.basicSalary}"


# Car class definition
class Car:
    # Constructor method for the Car class
    def __init__(self, id: int, name: str, price: int, type: str,
fuelCapacity: int, maxSpeed: int, color: str):
        self.id = id
        self.name = name
        self.price = price
        self.type = type
        self.fuelCapacity = fuelCapacity
        self.maxSpeed = maxSpeed
        self.color = color

    # Method to display the details of a Car
    def displayDetails(self) -> str:
        return f"ID: {self.id}, Name: {self.name}, Price: {self.price},
Type: {self.type}, Fuel Capacity: {self.fuelCapacity}, Max Speed:
{self.maxSpeed}, Color: {self.color}"

# Manager class definition (inherits from Employee)
class Manager(Employee):
    # Constructor method for the Manager class
    def __init__(self, id: int, name: str, dob: datetime.date, age: int,
passportDetails: str, department: str, jobTitle: str, basicSalary: int):
        super().__init__(id, name, dob, age, passportDetails, department,
jobTitle, basicSalary)
        self.salespersons: List[Salesperson] = []

    # Method to add a Salesperson to the Manager's list
    def addSalesperson(self, salesperson):
        self.salespersons.append(salesperson)

    # Method to remove a Salesperson from the Manager's list
    def removeSalesperson(self, salesperson):
        self.salespersons.remove(salesperson)

    # Method to display the details of a Manager (overrides Employee's
display_details)
    def displayDetails(self) -> str:
        return f"{super().displayDetails()}, Number of Salespersons:
{len(self.salespersons)}"

# Salesperson class definition (inherits from Employee)
class Salesperson(Employee):
    # Constructor method for the Salesperson class
    def __init__(self, id: int, name: str, dob: datetime.date, age: int,
passportDetails: str, department: str, jobTitle: str, basicSalary: int):
        super().__init__(id, name, dob, age, passportDetails, department,
jobTitle, basicSalary)
        self.sales: List[Sale] = []

    # Method to add a Sale to the Salesperson's list
    def addSale(self, sale):
        self.sales.append(sale)

    # Method to calculate commission for the Salesperson
    def calculateCommission(self) -> int:
        commission = 0
        for sale in self.sales:
            profit = sale.salePrice - sale.car.price
            commission += 0.065 * profit
        return commission

    # Method to display the details of a Salesperson (overrides Employee's
displayDetails)
    def displayDetails(self) -> str:
        return f"{super().displayDetails()}, Sales: {len(self.sales)},
Commission: {self.calculateCommission()}"

# Sale class definition
class Sale:
```

```python
    # Constructor method for the Sale class
    def __init__(self, salesperson, car, salePrice: int, saleDate:
datetime.date):
        self.salesperson = salesperson
        self.car = car
        self.salePrice = salePrice
        self.saleDate = saleDate
    # Method to display the details of a Sale
    def __str__(self):
        return f"Salesperson ID: {self.salesperson.id}, Car ID:
{self.car.id}, Sale Price: {self.salePrice}, Sale Date: {self.saleDate}"

# Function to save data to a file
def save_data_to_file(data, filename):
    with open(filename, 'wb') as file:
        pickle.dump(data, file)

# Function to load data from a file
def load_data_from_file(filename):
    try:
        with open(filename, 'rb') as file:
            data = pickle.load(file)
            return data
    except FileNotFoundError:
        return []

# Function to find an employee by their ID
def findEmployeeById(employees, id):
    for employee in employees:
        if employee.id == id:
            return employee
    return None

# Function to find a car by its ID
def findCarById(cars, id):
    for car in cars:
        if car.id == id:
            return car
    return None

# Function to display employee details by their ID
def displayEmployeeDetails(employees, id):
    employee = findEmployeeById(employees, id)
    if employee:
        return employee.displayDetails()
    else:
        return "Employee not found"

# Function to display car details by its ID
def displayCarDetails(cars, id):
    car = findCarById(cars, id)
    if car:
        return car.displayDetails()
    else:
        return "Car not found"

# Function to display sales details of a salesperson by their ID
def displaySalesDetails(employees, id):
    employee = findEmployeeById(employees, id)
    if employee and isinstance(employee, Salesperson):
        salesDetails = [str(sale) for sale in employee.sales]
        return "\n".join(salesDetails)
    else:
        return "Salesperson not found"

# Function to calculate salaries for a manager and their salespersons
def calculateSalaries(manager):
    manager_commission = 0
    for salesperson in manager.salespersons:
        for sale in salesperson.sales:
            profit = sale.salePrice - sale.car.price
            manager_commission += 0.035 * profit
    manager_salary = manager.basic_salary + manager_commission

    salespersons_salaries = {}
    for salesperson in manager.salespersons:
        commission = salesperson.calculate_commission()
        salespersons_salaries[salesperson.name] = salesperson.basic_salary
+ commission

    return manager_salary, salespersons_salaries


# Application class definition
class Application(tk.Tk):
    def __init__(self, employees, cars):
        super().__init__()
        self.title("Car Sales Management System")
        self.geometry("800x600")
        self.employees = employees
        self.cars = cars


        # Create widgets
        self.label_id = tk.Label(self, text="Enter ID:")
        self.entry_id = tk.Entry(self)
        self.btn_employee_details = tk.Button(self, text="Show Employee
Details", command=self.showEmployeeDetails)
        self.btn_car_details = tk.Button(self, text="Show Car Details",
command=self.showCarDetails)
        self.btn_sales_details = tk.Button(self, text="Show Sales Details",
command=self.showSalesDetails)
        self.text_output = tk.Text(self, wrap=tk.WORD)

        # Layout widgets
        self.label_id.grid(row=0, column=0, padx=5, pady=5, sticky=tk.W)
        self.entry_id.grid(row=0, column=1, padx=5, pady=5, sticky=tk.W)
        self.btn_employee_details.grid(row=1, column=0, padx=5, pady=5,
sticky=tk.W)
        self.btn_car_details.grid(row=1, column=1, padx=5, pady=5,
sticky=tk.W)
        self.btn_sales_details.grid(row=1, column=2, padx=5, pady=5,
sticky=tk.W)
        self.text_output.grid(row=2, column=0, columnspan=3, padx=5,
pady=5, sticky=tk.W + tk.E + tk.N + tk.S)

    # Method to show employee details when the "Show Employee Details"
button is clicked
    def showEmployeeDetails(self):
        id = int(self.entry_id.get())
        details = displayEmployeeDetails(self.employees, id)
        self.text_output.delete('1.0', tk.END)
```

```python
        self.text_output.insert(tk.END, details)

    # Method to show car details when the "Show Car Details" button is
clicked
    def showCarDetails(self):
        id = int(self.entry_id.get())
        details = displayCarDetails(self.cars, id)
        self.text_output.delete('1.0', tk.END)
        self.text_output.insert(tk.END, details)

    # Method to show sales details when the "Show Sales Details" button is
clicked
    def showSalesDetails(self):
        id = int(self.entry_id.get())
        details = displaySalesDetails(self.employees, id)
        self.text_output.delete('1.0', tk.END)
        self.text_output.insert(tk.END, details)

# Test cases
susan = Manager(47899, "Susan Meyers", datetime.date(1978, 6, 12), 44,
"123456789", "Accounting", "Manager", 37500)
mark = Salesperson(39119, "Mark Jones", datetime.date(1989, 4, 30), 34,
"987654321", "IT", "Salesperson", 26000)
joy = Salesperson(81774, "Joy Rogers", datetime.date(1992, 12, 8), 30,
"123987456", "Manufacturing", "Salesperson", 24000)

susan.addSalesperson(mark)
susan.addSalesperson(joy)

jazz = Car(1, "Jazz VX3", 55000, "Hatch", 40, 180, "Blue")
mark3 = Car(2, "Mark3 SX3", 84000, "Sedan", 50, 220, "Red")
wagoner = Car(3, "Wagoner ZX3", 125000, "SUV", 60, 240, "Black")

# Joy's sales
joy.addSale(Sale(joy, wagoner, 155000, datetime.date(2023, 5, 4)))
joy.addSale(Sale(joy, jazz, 57800, datetime.date(2023, 5, 5)))
joy.addSale(Sale(joy, jazz, 55000, datetime.date(2023, 5, 10)))
joy.addSale(Sale(joy, mark3, 89000, datetime.date(2023, 5, 15)))
joy.addSale(Sale(joy, mark3, 93000, datetime.date(2023, 5, 20)))

# Mark's sales
mark.addSale(Sale(mark, jazz, 58000, datetime.date(2023, 5, 3)))
mark.addSale(Sale(mark, jazz, 58000, datetime.date(2023, 5, 12)))
mark.addSale(Sale(mark, wagoner, 158000, datetime.date(2023, 5, 18)))
mark.addSale(Sale(mark, wagoner, 158000, datetime.date(2023, 5, 22)))
mark.addSale(Sale(mark, wagoner, 158000, datetime.date(2023, 5, 25)))

print(susan.displayDetails())
print(mark.displayDetails())
print(joy.displayDetails())

print(jazz.displayDetails())
print(mark3.displayDetails())
print(wagoner.displayDetails())


if __name__ == "__main__":
    employees = [susan, mark, joy]
    cars = [jazz, mark3, wagoner]
    app = Application(employees, cars)
    app.mainloop()
```

Summary:

Throughout the entire assignment, I've explored the implementation of a Car Sales Management System in Python. This system demonstrates key object-oriented programming concepts, such as inheritance, composition, aggregation, and association, through classes like Person, Employee, Salesperson, and Car. I've also learned about using pickle for data storage and retrieval and tkinter for building a basic GUI. Furthermore, I've gained an understanding of how to represent these relationships using UML diagrams, including multiplicities. I've also examined the use of Python's built-in datetime module for handling date-related information, the enum module for defining enumeration types like CarType. I’ve explored the functionality of the system, such as calculating commissions for salespersons, storing, and displaying employee and car details, as well as sales data. I have seen how the system effectively models real-world relationships, such as the association between a Manager and their Salespersons, and the link between a Car and a Sale.
Overall, this assignment has let me show my understanding and apply various Python features and object-oriented programming concepts to create a functional Car Sales Management System.

Explanation:

Classes:

1. CarType: An enumeration class to define car types.
2. Person: A class for a general person with attributes like id, name, date of birth, age, and passport details.
3. Employee: A subclass of Person representing employees with additional attributes like department, job title, and basic salary.
4. Car: A class for cars with attributes like id, name, price, type, fuel capacity, max speed, and color.
5. Manager: A subclass of Employee representing managers. It has an additional attribute, salespersons, to store the list of Salespersons managed by the manager.
6. Salesperson: A subclass of Employee representing salespersons. It has an additional attribute, sales, to store the list of sales made by the salesperson.
7. Sale: A class for sales with attributes like salesperson, car, sale price, and sale date.
8. Application: A tkinter-based GUI application class that allows users to view employee, car, and sales details.
    a. GUI Application: The Application class extends the tkinter Tk class to create a GUI for the Car Sales Management System. The GUI includes input fields, buttons, and a text area to display the information requested by the user.
    b. Button commands: The Application class has methods (showEmployeeDetails, showCarDetails, and showSalesDetails) that are executed when their corresponding buttons are clicked. These methods use the utility functions to fetch and display the relevant information in the text area.
9. Button commands: The Application class has methods (showEmployeeDetails, showCarDetails, and showSalesDetails) that are executed when their corresponding buttons are clicked. These methods use the utility functions to fetch and display the relevant information in the text area.

When a user interacts with the GUI, they can enter an ID and click the appropriate button to view employee details, car details, or sales details. The application will get the relevant information using the utility functions and display it in the text area.

Github: https://github.com/alyalootah/Final