from abc import ABC, abstractmethod

# Abstract Vehicle Class (Abstraction)

class Vehicle(ABC):

def \_\_init\_\_(self, vehicle\_id, model, base\_rate):

self.\_\_vehicle\_id = vehicle\_id

self.\_\_model = model

self.\_\_base\_rate = base\_rate

self.\_\_is\_available = True

@abstractmethod

def calculate\_rental\_cost(self, days):

pass

@abstractmethod

def is\_available\_for\_rental(self):

return self.\_\_is\_available

def get\_vehicle\_details(self):

return {

"Vehicle ID": self.\_\_vehicle\_id,

"Model": self.\_\_model,

"Base Rate": self.\_\_base\_rate,

"Available": self.\_\_is\_available

}

def set\_availability(self, status):

if isinstance(status, bool):

self.\_\_is\_available = status

# Inherited Classes (Inheritance)

class Car(Vehicle):

def \_\_init\_\_(self, vehicle\_id, model, base\_rate, seating\_capacity):

super().\_\_init\_\_(vehicle\_id, model, base\_rate)

self.\_\_seating\_capacity = seating\_capacity

def calculate\_rental\_cost(self, days):

return self.get\_vehicle\_details()["Base Rate"] \* days \* 1.1 # 10% extra for maintenance

def is\_available\_for\_rental(self):

return super().is\_available\_for\_rental()

class Motorcycle(Vehicle):

def \_\_init\_\_(self, vehicle\_id, model, base\_rate, engine\_capacity):

super().\_\_init\_\_(vehicle\_id, model, base\_rate)

self.\_\_engine\_capacity = engine\_capacity

def calculate\_rental\_cost(self, days):

return self.get\_vehicle\_details()["Base Rate"] \* days \* 0.9 # Discount for light maintenance

def is\_available\_for\_rental(self):

return super().is\_available\_for\_rental()

class Truck(Vehicle):

def \_\_init\_\_(self, vehicle\_id, model, base\_rate, load\_capacity):

super().\_\_init\_\_(vehicle\_id, model, base\_rate)

self.\_\_load\_capacity = load\_capacity

def calculate\_rental\_cost(self, days):

return self.get\_vehicle\_details()["Base Rate"] \* days \* 1.5 # Heavy maintenance surcharge

def is\_available\_for\_rental(self):

return super().is\_available\_for\_rental()

# Interface and Polymorphism

class Rentable:

def rent(self, customer, days):

pass

def return\_vehicle(self):

pass

class RentalCar(Car, Rentable):

def rent(self, customer, days):

if self.is\_available\_for\_rental():

self.set\_availability(False)

return self.calculate\_rental\_cost(days)

return "Car not available"

def return\_vehicle(self):

self.set\_availability(True)

class RentalMotorcycle(Motorcycle, Rentable):

def rent(self, customer, days):

if self.is\_available\_for\_rental():

self.set\_availability(False)

return self.calculate\_rental\_cost(days)

return "Motorcycle not available"

def return\_vehicle(self):

self.set\_availability(True)

class RentalTruck(Truck, Rentable):

def rent(self, customer, days):

if self.is\_available\_for\_rental():

self.set\_availability(False)

return self.calculate\_rental\_cost(days)

return "Truck not available"

def return\_vehicle(self):

self.set\_availability(True)

# Supporting Classes (Composition)

class Customer:

def \_\_init\_\_(self, customer\_id, name):

self.customer\_id = customer\_id

self.name = name

class RentalTransaction:

def \_\_init\_\_(self, customer, vehicle, days):

self.customer = customer

self.vehicle = vehicle

self.days = days

self.cost = vehicle.calculate\_rental\_cost(days)

class RentalAgency:

def \_\_init\_\_(self, name):

self.name = name

self.\_\_vehicles = []

self.\_\_transactions = []

def add\_vehicle(self, vehicle):

self.\_\_vehicles.append(vehicle)

def list\_vehicles(self):

return [vehicle.get\_vehicle\_details() for vehicle in self.\_\_vehicles]

def rent\_vehicle(self, customer, vehicle\_id, days):

for vehicle in self.\_\_vehicles:

if vehicle.get\_vehicle\_details()["Vehicle ID"] == vehicle\_id:

if vehicle.is\_available\_for\_rental():

transaction = RentalTransaction(customer, vehicle, days)

self.\_\_transactions.append(transaction)

vehicle.set\_availability(False)

return transaction.cost

else:

return "Vehicle not available"

return "Vehicle not found"

def return\_vehicle(self, vehicle\_id):

for vehicle in self.\_\_vehicles:

if vehicle.get\_vehicle\_details()["Vehicle ID"] == vehicle\_id:

vehicle.set\_availability(True)

return "Vehicle returned successfully"

return "Vehicle not found"

# Example Usage

if \_\_name\_\_ == "\_\_main\_\_":

# Creating the rental agency and adding vehicles

agency = RentalAgency("Elite Rentals")

car1 = RentalCar("C123", "Toyota Corolla", 50, 5)

motorcycle1 = RentalMotorcycle("M456", "Yamaha MT", 20, 500)

truck1 = RentalTruck("T789", "Ford F150", 100, 2000)

agency.add\_vehicle(car1)

agency.add\_vehicle(motorcycle1)

agency.add\_vehicle(truck1)

# Customer renting a vehicle

customer1 = Customer("CU001", "John Doe")

rental\_cost = agency.rent\_vehicle(customer1, "C123", 3)

print(f"Rental cost for John Doe: {rental\_cost}")

# Listing available vehicles

print("\nAvailable vehicles:")

for vehicle in agency.list\_vehicles():

print(vehicle)

# Returning a vehicle

agency.return\_vehicle("C123")

print("\nVehicle returned. Updated list:")

for vehicle in agency.list\_vehicles():

print(vehicle)