from abc import ABC, abstractmethod

from datetime import datetime

# Abstract Base Class for Vehicle

class Vehicle(ABC):

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

self.\_vehicle\_id = vehicle\_id

self.\_brand = brand

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\_info(self):

return f"ID: {self.\_vehicle\_id}, Brand: {self.\_brand}, Model: {self.\_model}"

def set\_availability(self, availability):

if not isinstance(availability, bool):

raise ValueError("Availability must be a boolean.")

self.\_is\_available = availability

# Rentable Interface

class Rentable:

def rent(self, customer, days):

pass

def return\_vehicle(self):

pass

# Concrete Vehicle Classes

class Car(Vehicle, Rentable):

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

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

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

return self.\_base\_rate \* days

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

return self.\_is\_available

def rent(self, customer, days):

if not self.is\_available\_for\_rental():

raise Exception("Car is not available for rental.")

self.set\_availability(False)

return f"Car rented to {customer.name} for {days} days."

def return\_vehicle(self):

self.set\_availability(True)

return "Car returned successfully."

class Motorcycle(Vehicle, Rentable):

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

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

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

discount\_rate = 0.9 if days > 7 else 1

return self.\_base\_rate \* days \* discount\_rate

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

return self.\_is\_available

def rent(self, customer, days):

if not self.is\_available\_for\_rental():

raise Exception("Motorcycle is not available for rental.")

self.set\_availability(False)

return f"Motorcycle rented to {customer.name} for {days} days."

def return\_vehicle(self):

self.set\_availability(True)

return "Motorcycle returned successfully."

class Truck(Vehicle, Rentable):

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

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

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

return self.\_base\_rate \* days + 50 # Additional fixed charge

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

return self.\_is\_available

def rent(self, customer, days):

if not self.is\_available\_for\_rental():

raise Exception("Truck is not available for rental.")

self.set\_availability(False)

return f"Truck rented to {customer.name} for {days} days."

def return\_vehicle(self):

self.set\_availability(True)

return "Truck returned successfully."

# Supporting Classes

class Customer:

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

self.customer\_id = customer\_id

self.name = name

class RentalTransaction:

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

self.transaction\_id = transaction\_id

self.customer = customer

self.vehicle = vehicle

self.days = days

self.start\_date = datetime.now()

self.end\_date = self.start\_date.replace(day=self.start\_date.day + days)

def get\_transaction\_details(self):

return (

f"Transaction ID: {self.transaction\_id}\n"

f"Customer: {self.customer.name}\n"

f"Vehicle: {self.vehicle.get\_vehicle\_info()}\n"

f"Duration: {self.days} days\n"

f"Cost: ${self.vehicle.calculate\_rental\_cost(self.days)}"

)

class RentalAgency:

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

self.name = name

self.vehicles = []

self.customers = []

def add\_vehicle(self, vehicle):

if not isinstance(vehicle, Vehicle):

raise ValueError("Only Vehicle instances can be added.")

self.vehicles.append(vehicle)

def add\_customer(self, customer):

if not isinstance(customer, Customer):

raise ValueError("Only Customer instances can be added.")

self.customers.append(customer)

def find\_available\_vehicle(self, vehicle\_type):

for vehicle in self.vehicles:

if isinstance(vehicle, vehicle\_type) and vehicle.is\_available\_for\_rental():

return vehicle

raise Exception("No available vehicles of the requested type.")