Assignment DCIT 201

ID NUMBER: 22052103

1. ABSTRACTION

from abc import ABC, abstractmethod

class Vehicle (ABC):

def \_\_in it \_\_(self, vehicle id, brand, model, year, rental price):

self . \_\_vehicle id = vehicle id

self. brand = brand

self. model = model

self. year = year

self. rental price = rental price

@abstractmethod

def calculate rental cost(self, days):

pass

@abstractmethod

def is available for rental(self):

pass

def get vehicle id(self):

return self.\_ vehicle id

def get brand(self):

return self. brand

def get model(self):

return self. model

def get year(self):

return self. year

def get rental price(self):

return self. rental price

1. INHERITANCE

class Car(Vehicle):

def \_\_init\_\_(self, vehicle id, brand, model, year, rental price, num doors):

super().\_\_init\_\_(vehicle id, brand, model, year, rental price)

self.\_ num doors = num doors

def calculate rental cost(self, days):

return self. get rental price() \* days

def is available for rental(self):

# Implement availability logic

return True

def get num doors(self):

return self. num doors

class Motorcycle(Vehicle):

def \_\_init\_\_(self, vehicle id, brand, model, year, rental price, has sidecar):

super().\_\_init\_\_(vehicle id, brand, model, year, rental price)

self. has sidecar = has sidecar

def calculate rental cost(self, days):

return self.get rental price() \* days \* 0.9 # 10% discount for motorcycles

def is available for rental(self):

# Implement availability logic

return True

def get has sidecar(self):

return self. has sidecar

class Truck(Vehicle):

def \_\_init\_\_(self, vehicle id, brand, model, year, rental price, max load capacity):

super().\_\_init\_\_(vehicle id, brand, model, year, rental price)

self. max load capacity = max load capacity

def calculate rental cost(self, days):

return self. get rental price() \* days \* 1.2 # 20% surcharge for trucks

def is available for rental(self):

# Implement availability logic

return True

def get max load capacity(self):

return self. max load capacity

1. ENCAPSULATION

class Vehicle(ABC):

def \_\_init\_\_(self, vehicle id, brand, model, year, rental price):

self. vehicle id = vehicle id

self. brand = brand

self. model = model

self.\_ year = year

self.\_ rental price = rental price

def get vehicle id(self):

return self.\_ vehicle id

def get brand(self):

return self.\_ brand

def get model(self):

return self.\_ model

def get year(self):

return self.\_ year

def get rental price(self):

return self.\_ rental price

def set rental price(self, rental price):

if rental price > 0:

self.\_ rental price = rental price

else:

raise ValueError("Rental price must be positive")

class Car(Vehicle):

def \_\_init\_\_(self, vehicle id, brand, model, year, rental price, num doors):

super().\_\_init\_\_(vehicle id, brand, model, year, rental price)

self.\_ num doors = num doors

def get num doors(self):

return self. num doors

def set num doors(self, num doors):

if num doors > 0:

self.\_ num doors = num doors

else:

raise ValueError("Number of doors must be positive")

def calculate rental cost(self, days):

return self. get rental price() \* days

def is available for rental(self):

return True

1. POLYMORPHISM

class Rentable(ABC):

@abstractmethod

def rent(self, customer, days):

pass

@abstractmethod

def return vehicle(self):

pass

class Car(Vehicle, Rentable):

def \_\_init\_\_(self, vehicle id, brand, model, year, rental price, num doors):

super().\_\_init\_\_(vehicle id, brand, model, year, rental price)

self.\_ num doors = num doors

def rent(self, customer, days):

print(f"Car rented to {customer get name()} for {days} days")

def return vehicle(self):

print("Car returned")

def calculate rental cost(self, days):

return self.get rental price() \* days

def is available for rental(self):

return True

1. COMPOSITION

class Customer:

def \_\_init\_\_(self, customer id, name, email):

self.\_ customer id = customer id

self.\_ name = name

self.\_ email = email

def get customer id(self):

return self.\_ customer id

def get name(self):

return self.\_ name

def get email(self):

return self.\_ email

class RentalTransaction:

def \_\_init\_\_(self, transaction id, customer, vehicle, rental date, return date):

self.\_ transaction id = transaction id

self.\_ customer = customer

self.\_ vehicle = vehicle

self.\_ rental date = rental date

self.\_ return date = return date

def get\_transaction\_details(self):

return {

"transaction\_id": self.\_\_transaction\_id,

"customer": self.\_\_customer.get\_name(),

"vehicle": self.\_\_vehicle.get\_model(),

"rental\_date": self.\_\_rental\_date,

"return\_date": self.\_\_return\_date

}

class RentalAgency:

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

self.\_\_name = name

self.\_\_vehicles = []

self.\_\_customers = []

def add\_vehicle(self, vehicle):

self.\_\_vehicles.append(vehicle)

def add\_customer(self, customer):

self.\_\_customers.append(customer)

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

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

vehicle.rent(customer, days)

transaction = RentalTransaction(len(self.\_\_vehicles), customer, vehicle, "2024-12-01", "2024-12-10")

print(transaction.get\_transaction\_details())

else:

print("Vehicle not available for rental")