JIT KWONG CHEO

T1A3 - Terminal App Presentation

Features of my ParkingLot Terminal App & it's usage.

Features & usages:

- Taking user's input for vehicle type and registration numbers.
- Allocate parking spaces availability based on the number of parking spaces.
- Calculate parking fee for the vehicle based on vehicle entry & exit time.
- Generate report on vehicle registration & type, entry & exit time as well as parking fees.

Review of development/build process(challenges & favourite parts)

Challenges:

- Trial & error in getting the report to generate the right parking fee calculation & parking time.
- Coming up with the correct testing procedure on testing my ParkingLot app to make sure that it passes the require test through pytest.

Favourites:

- Being able to see user to enter vehicle type and registration & parking spaces being allocated.
- Seeing the generated report with the correct information.

Walkthrough logic of my parking ParkingLot Terminal App

Class Vehicle:

```python
import time

class Vehicle:
    def __init__(self, v_type, plate, entry_time=None, exit_time=None, fee=None, entry_datetime=None, exit_datetime=None):
        self.type = v_type # vehicle type
        self.plate = plate # registration number
        self.entry_time = entry_time if entry_time is not None else time.time()
        # entry_time indicates time vehicle enter the lot,
        # not provided it defaults to the time imported.
        self.exit_time = exit_time # exit time
        self.fee = fee # stores parking fee, set to None
        self.entry_datetime = entry_datetime
        # additional attributes to store them as datetime objects.
        self.exit_datetime = exit_datetime
```

Class ParkingLot:

```python
import time
from vehicle import Vehicle

class ParkingLot:
    def __init__(self, total_spaces):
        # total_spaces shows # space in the lot.
        # spaces is list that represent parking spaces.
        # spaces are set to none to indicate they are not occupied.
        self.spaces = [None] * total_spaces
        #available_spaces represent available space is in the lot.
        self.available_spaces = total_spaces
        # occupied_spaces represent number spaces being occupied.
        self.occupied_spaces = 0
```

Class ParkingLot:

```
def park_vehicle(self, v_type, plate):
    # no parking spaces available, print the error code
    if self.available_spaces == 0:
        print("Error: No Available Spaces")
        return

    # below code proceeds to check for space as long as code pass above
    for i in range(len(self.spaces)):
      if self.spaces[i] is None:
            # import time module set the time for time & datetime
            current_time = time.time()
            current_datetime = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
            # creates the vehicle as per the arguments in the ()
            vehicle = Vehicle(v_type, plate, entry_time=current_time, entry_datetime=current_datetime)
            self.spaces[i] = vehicle
            # depending on vehicle entering & exiting,
            # below 2 spaces are updated accordingly.
            self.available_spaces -= 1
            print("Vehicle parked successfully.")
            self.occupied_spaces += 1
            return vehicle

    # if parking is full, below error will be printed.
    print("Error: Parking lot is full")
    return None
```

Class ParkingLot:

```python
# function that take parameter 'plate'/registration plate.
def exit_lot(self, plate):
    for i in range(len(self.spaces)):
        # inside this nested loops, if the plate matches,
        if self.spaces[i] and self.spaces[i].plate == plate:
            # retrieve the time
            current_time = time.time()
            # change the format to YYY/MM/DD HH:MM:SS
            current_datetime = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
            vehicle = self.spaces[i]
            vehicle.exit_time = current_time
            vehicle.exit_datetime = current_datetime
            # then calculate the fee and store it as 'fee'
            fee = self.calculate_fee(vehicle)
            self.spaces[i] = None
            self.available_spaces += 1
            self.occupied_spaces -= 1
            return vehicle, fee

    # # Vehicle not found and return none for the vehicle object & fee
    return None, None
```

Class ParkingLot:

```
# display status of the parkinglot
def display_lot(self):
    print("Available spaces:", self.available_spaces)
    print("Parking Lot Status:")
    for i in range(len(self.spaces)):
        # print the following if space is occupied
        if self.spaces[i]:
            print(f"Space {i+1}: {self.spaces[i].plate} ({self.get_vehicle_type(self.spaces[i].type)})")
        else:
            print(f"Space {i+1}: Empty")
```

Class ParkingLot:

```python
# fee calculations
def calculate_fee(self, vehicle):
    # this method check if the time is noe, means is not recorded
    if vehicle.entry_time is None or vehicle.exit_time is None:
        # it will then indicate that fee can't be calc.
        return None

    # if enter & exit time are recorded, substraction happens.
    duration = vehicle.exit_time - vehicle.entry_time
    # Assuming the fee calculation is $4 per hour
    fee = duration * 4 / 3600  # Convert seconds to hours
    vehicle.fee = fee
    # calculated fee is processed.
    return fee
```

Class ParkingLot:

```
# belongs to the class rather than instance of class.
@staticmethod
def get_vehicle_type(v_type):
    if v_type == 1:
        return "Car"
    elif v_type == 2:
        return "Truck"
    elif v_type == 3:
        return "Motorcycle"
    else:
        return "Unknown"
```

main.py

```python
import time
from vehicle import *
from parkinglot import ParkingLot

def main():
    # user input number of parking spaces
    total_spaces = int(input("Enter the total number of parking spaces: "))
    parking_lot = ParkingLot(total_spaces)
    # List to store the report entries
    report = []

    # while loop to keep repeating until quit is selected.
    while True:
        print("\nPlease select an option:")
        print("1. Park a vehicle")
        print("2. Exit the parking lot")
        print("3. Display parking lot status")
        print("4. Quit")

        option = input("> ")
```

main.py

```
if option == "1":
    print("\nSelect a vehicle type:")
    print("1. Car")
    print("2. Truck")
    print("3. Motorcycle")

    v_type = int(input("> "))
    plate = input("Enter vehicle registration number: ")
    vehicle = parking_lot.park_vehicle(v_type, plate)
    if vehicle is not None:
        # if vehicle is parked, report on information of v_plate, entry & exit time & fee will
        # be generated.
        report.append(f"Parked: {vehicle.plate} ({parking_lot.get_vehicle_type(vehicle.type)}), "
                      f"Entry: {vehicle.entry_datetime}, Fee: ${vehicle.fee}")


elif option == "2":
    plate = input("Enter vehicle registration number: ")
    # 2 arguments as vehicle & fee
    vehicle, fee = parking_lot.exit_lot(plate)
    if vehicle is not None:
        if fee is not None:
            # if vehicle & fee returns correct value,
            print(f"Vehicle exited successfully. Fee: ${fee:.2f}")
            # generate repprt on exiting infos
            report.append(f"Exited: {vehicle.plate} ({parking_lot.get_vehicle_type(vehicle.type)}), "
                          f"Exit: {vehicle.exit_datetime}, Fee: ${fee:.2f}")
        else:
            print("Error: Fee calculation failed.")
```

main.py

```
        # display status of the parking lot whether is full or empty
        elif option == "3":
            parking_lot.display_lot()

        # option 4 is quitting the terminal app
        elif option == "4":
            # Generate the report and write it to a file
            # strftime format the date to YYY-MM-DD_HH:MM:SS
            timestamp = time.strftime("%Y-%m-%d_%H-%M-%S", time.localtime())
            filename = f"parking_report_{timestamp}.txt"

            # file open in written mode, 'with' will ensure that file is properly closed
            with open(filename, "w") as file:
                # loop iterates the report, entry report is written with a newline
                for entry in report:
                    file.write(entry + "\n")

            # success message printed indicated report generated & saved.
            print(f"Report generated and saved as {filename}.")
            break

        else:
            print("Invalid option. Please try again.")


# using the same code as original repo,
# from reading, check current script run directly or imported
# Way to ensure certain code runs only when script is run direxctly,
# not when is imported as module.
if __name__ == '__main__':
    main()
```

Walkthrough logic of my parking ParkingLot Terminal App

test_main.py

```python
import pytest
import time
from parkinglot import ParkingLot
from vehicle import Vehicle


#pytest fixture, function that can used to manage apps states & dependencies
@pytest.fixture
def create_vehicle():
    # to get current time
    current_time = time.time()
    # initialize vehicle as car with plate of ABC123 with a current time
    return Vehicle(1, "ABC123", entry_time=current_time)


@pytest.fixture
def parking_lot():
    # capacity of 10 parkinglot
    return ParkingLot(10)


def test_vehicle_creation(create_vehicle):
    vehicle = create_vehicle
    # assertion to verify attributes matches expected values.
    assert vehicle.type == 1
    assert vehicle.plate == "ABC123"

# check if entry_time is either None or valid time value
def test_entry_time(create_vehicle):
    # create_vehicle obtain 'Vehicle' object name 'vehicle'
    vehicle = create_vehicle
    # time.time() is to get current time and store in it.
    current_time = time.time()
    assert vehicle.entry_time is None or vehicle.entry_time <= current_time
```

test_main.py

```python
# check if entry_time can be set to specific value and retains the value correctly.
def test_set_entry_time(create_vehicle):
    vehicle = create_vehicle
    current_time = time.time()
    # # to represent a time that is an hours earlier than current time
    vehicle.entry_time = current_time - 3600
    assert vehicle.entry_time == current_time - 3600


def test_calculate_fee():
    # parking capacity of 10
    parking_lot = ParkingLot(10)
    # representing time vehicle entered the parkinglot
    entry_time = time.time() - 7200  # Two hours ago
    exit_time = time.time()  # Current time
    # vehicle parked 2 hours
    vehicle = Vehicle(1, "ABC123", entry_time=entry_time, exit_time=exit_time)
    parking_lot.spaces[0] = vehicle
    # to calc parking fee based on duration of vehicle parking
    fee = parking_lot.calculate_fee(vehicle)
    assert fee == 8  # Assuming $4 per hour, the fee for 2 hours should be $8

def test_park_vehicle(create_vehicle, parking_lot):
    vehicle = create_vehicle
    # Park the vehicle
    result = parking_lot.park_vehicle(vehicle.type, vehicle.plate)
    # Verify that the vehicle is parked successfully
    assert result is not None
    # is not None indicate vehicle parked successfully
    assert result.plate == vehicle.plate
    # to make sure the correct vehicle is parked

    # indicate 1 parking space is now occupied
    assert parking_lot.occupied_spaces == 1
```