Programming BI. Part 2
Homework assignment 1.
Smart parking application

1 Tested skills

- Basic OOP and class composition/aggregation
- Collections
- File input-output

2 Introduction

In the first homework assignment of the C# programming course you will implement a back-end[1] of a smart parking system for a large shopping centre.

[1]The term back-end relates to the functionality of an application that the end user does not see, however it is essential for proper functioning of the whole system

The system under consideration can serve as the central interface between gate control systems with car plate recognition devices, parking payment kiosks, mobile applications for users and other system components.

Let's first describe a typical parking system from the user's perspective:

1. A visitor arrives at one of the entry gates, a printed ticket is issued and the car is allowed in, provided there are free parking places. In our smart parking system there is a video recognition system that reads the car plate number on entrance and on exit from the parking.

2. The visitor parks the car at a free spot.

3. After leaving the shopping centre, the client inserts the parking ticket to a payment kiosk and pays for the parking session according to the tariff table.

   The tariff table is supposed to be of the following format (actual numbers may differ):

Minutes	Rate (RUB)
15	0
60	50
120	100
180	140
240	180
...	
600+	350

   The tarification policy in the table above should be understood as follows: parking sessions of up to 15 minutes long are free, sessions of up to 60 minutes long cost 50 RUB, etc. Sessions lasting more than 600 minutes are tarified according to the maximal rate (last row of the table). In the assignment we will not consider the difficult cases of visitors leaving their cars for more than a day. In any case just take the difference between the current time and the entry time (in minutes), even if one year passed between the two, you will still charge 350 RUB.

4. After making the payment, the visitor is granted additional time to leave the parking, we will refer to this as "free exit period". To make this logically aligned with the tariff table, the free exit period can be taken as the value from the first line (15 minutes in the table above), this period is counted from the payment time, not from the entry time!

5. On leaving the parking, the visitor inserts the ticket at the exit gate to a special machine, if the car leaves within the allowed free time after payment, the gate opens. Otherwise, exit is denied and the client has to pay additional charge according to the tariff table.

Below are some examples (it is assumed that all times are within the same day, the tariff table from above is taken for calculation):

Scenario 1:

1. Car enters the parking at 10:05:10

2. Payment is performed at 11:05:11, charge = 100 RUB

3. Car exits the parking at 11:10:00, exit is granted

Scenario 2:

1. Car enters the parking at 12:10:00

2. Payment is performed at 12:16:45, charge = 0 RUB

3. Car exits the parking at 12:30:01, exit is granted

Scenario 3:

1. Car enters the parking at 20:40:06
2. Payment is performed at 22:00:03, charge = 100 RUB
3. Car attempts to exit the parking at 22:30:05, exit is denied (more than 15 minutes passed since 22:00:03)
4. Additional payment is performed at 22:33:10, extra charge for the period 22:00:03-22:33:10 = 50 RUB (total charge for the session = 150 RUB)
5. Car exits the parking at 22:40:00, exit is granted

In the advanced task you will also implement one improvement to the system that will allow to register user accounts and perform automatic payments (possibly from the connected credit cards or bonus cards) without having to insert parking tickets to a payment kiosk.

In this case will automatically link the car plate number from an image recognition system on exit and link it to an active parking session (more detailed description is provided in the respective section of the assignment)

3 Project template description

Complete your work in the supplied console application, don't create a new project!

Don't change names of classes, methods or properties that are already given to you. You are free to choose the names of new program elements that you will be asked to add.

The template contains definitions of classes forming the model of the subject area (ParkingSession, Tariff, User) and the central class, which you need to implement - ParkingManager.

4 Description of tasks

Basic task - 5 points

For the basic task you won't need the **User** class.

- Study the two classes - ParkingSession and Tariff - with their properties provided in the template. The "Dt" suffix in several properties is one of the common ways to specify that the property is a DateTime. Before moving on, understand the logical connection of these classes to the description of the subject area above. Find the explanation of why certain properties in the ParkingSession class are nullable, while the others aren't.

  Remarks:

  - The **PaymentDt** property should store the date and time of the last payment (see scenario 3 in the previous section)
  - The **TotalPayment** property should store the total amount from all payments made for the parking

- Declare the following collections as private fields inside the ParkingManager class:

  - List of active parking sessions
  - List of completed parking sessions
  - Tariff table. This should represent a table of the same structure as mentioned in the introduction (you may fill the table with your own numbers)

- Declare two additional fields/properties inside the ParkingManager class that will be used in the algorithms: ParkingCapacity (total number of places) and FreeLeavePeriod (this can be taken from the first line of the tariff table as described above)
- Implement the following methods of the ParkingManager class:
  - EnterParking
  - TryLeaveParkingWithTicket
  - GetRemainingCost
  - PayForParking

  The logic that each method should follow is specified in the comments inside the code.

  When implementing the algorithms, you may find that certain additional data structures may be useful. If you use such data structures in a proper way, this will give you a bonus.
- Write any logic in the main program that will check operation of the methods implemented above. You can supply your code with comments that explain how exactly you verify operation of the ParkingManager class and its methods in different scenarios, especially if it involves using the debugging mode.

Additional task 1: data persistence - 3 points

Add methods to the ParkingManager class to persist the entire dataset (sessions, tariffs, parking capacity) of the application to a file and load it back on startup. You are free to choose the exact approach (manual file IO / serialization), number of files (single file for everything or separate files for each dataset) and the file format (text, binary).

Note that the file should be resaved on each change to the application data.

Extend the testing logic from the basic part if necessary.

Additional task 2: simple loyalty system - 2 points

Add the following features to your ParkingManager class:

- Add a collection of registered visitors (users). This should also be loaded from a file. The registration process is not part of this assignment, you can simply predefine the list in the file (however, it should not be hardcoded)
- Add the following logic to the "EnterParking" method: when creating a new parking session, check whether there is a registered user with the same car plate number. If such user exists, link the parking session to the user object (you will need to adjust the ParkingSession class). When saving data, make sure that the link between a session and a user is also saved.
- Implement the TryLeaveParkingByCarPlateNumber method, detailed description is provided in the method's comments in the template.
- Extend the testing logic from the basic part if necessary.

5 Submission

Submit your solution following these steps:

1. Delete two temporary folders - "bin" and "obj" from the project folder.
2. Add the whole solution folder to a ZIP **(not RAR or 7Z)** archive.
3. Upload the archive to the Canvas LMS in the corresponding section

6 Grading policy

In general, the grade is determined by the quantity and quality of completed tasks taking into account the maximal possible marks for each of the three parts. The overall grade can be lowered in the following cases:

- Inefficient implementation of an algorithm (-1 point)
- Information duplicated multiple times in the file (-1 point)
- Poor programming style (-1 point)