# Additional assumptions and explanations

## Assumptions

- Possible currencies are stored in database - at least one has to be added to charge drivers for parking,
- Possible driver types are stored in database – it can be *regular*, *disabled* and other if needed in the future. It is also necessary to add *regular* and *disabled* types to charge drivers the way that it was described in the task,
- After stopping parking meter, driver is desired to pay for parking. Drivers' payments are stored in database to allow parking owner calculate earnings (e.g for a given day – as mentioned in the task)

## Use cases in action

1. **As a driver, I want to start the parking meter, so I don't have to pay the fine for the invalid parking.**

   Driver sends *post request* ("/api/start"). The body of the request has to contain driver's JSON object representation. For instance: driver uses mobile app where he has to be registered to use its functions (*registered* means that app has access to his *driver* object which has specified *licensePlate* and *DriverType* properties). He chooses *start* option and app send appropriate request with aforementioned JSON.
   Parking meter app (server) returns JSON representation of *ParkAction* object. It contains information about:

   - When driver started parking meter
   - Driver object (his licensePlate, driverType)
   - When driver stopped parking meter (after starting, it will be *null)*

2. **As a parking operator, I want to check if the vehicle has started the parking meter.**

   Parking operator sends *get request ("/api/check/licensePlate")* where *licensePlate* is licensPlate of driver that he wants to check.
   Parking meter app returns *true/false* value depending on whether driver started parking meter or not.

3. **As a driver, I want to stop the parking meter, so that I pay only for the actual parking time.**

   Driver sends *put request("/api/stop")* with *driver object JSON representation* in the body of request.

   Parking meter app returns JSON representation of list of *DriverCharge* objects. Each one is basically a fee that driver should pay in different currencies. App assumpts that driver can choose one currency and pay with it *("/api/pay").*

4. **As a driver, I want to know how much I have to pay for parking.**
   Driver (or parking operator who will also be able to do this) sends *get request* (either *"/api/charge" with driver object JSON representation* as body of request *or "/api/charge/licensePlate"* where *licensePlate* is driver's licensePlate).
   Parking meter app returns JSON representation of list of *DriverCharge* objects.

5. **As a parking owner, I want to know how much money I earn any given day.**
   Parking owner sends *get request ("/api/earnings/date")* where *date* is textual representation of a day that he wants calculate earnings for. The date has to be in format "yyyy-mm-dd", e.g. 2018-12-06.
   Parking meter app returns JSON representation of list of *DriverCharge* objects. Each one is calculated amount of money in every available currency.

## Classes which usage or existence may be found unobvious

- ***DriverCharge***
  This class represents fee and currency pair. It will be useful in a situation when parking owner wants to give different charges for different currencies. Exact calculation from one currency to another may be unfriendly to potential drivers.

- ***DriverPayment***
  This class represents details about driver payment for parking. It can be considered as one *DriverCharge* that has been paid by driver. It extends *DriverCharge* class and adds more fields (*payDate* and *id* which can be primary key in database)

- ***ParkingCost***
  This class represents prices for parking in one Currency. Contains fields:
  - *Currency* – currency related to prices
  - *firstHourValue* – fee for first hour of parking
  - *secondHourValue* – fee for second hour of parking
  - *overTwoHoursMultiplier* – represents value to multiply previous hour cost by when calculating cost of next hour of parking. For example:
    *thirsHourCost = secondHourValue * overTwoHoursMultiplier*

  Difference between *ParkingCost* and *DriverCharge* – *DriverCharge* is related to already calculated fee for parking for given driver and *ParkingCost* is related to general rules of calculating fees for parking. Each *DriverType* object should have another *ParkingCost*property (e.g. *disabled* driver pays another amounts of money for parking than *regular* driver does).

- **FakeDatabaseStub**
  This class represents faked database. Contains static list of objects that could be stored in database and are needed for the task to be complete.

## Decisions made to keep things simple

- In service layer, many methods can return null if something goes wrong. In such situations, some exception could be thrown, but I decided that it would be more convenient to check null value while using such methods.
- There is no *DriverTypeDAO* interface (nor class implementing it, of course). It is not necessary in this app (to do the given task correctly), as database is mocked and any of classes in service layer uses it. If desired, it can be added in the future very easily.

## Additional functionalities

- Error handling – if bad request is sent to app, JSON representation of *ParkingMeterErrorResponse* object is returned.
  Such object contains:
  - Message – message of exception (error)
  - Status – http response status
  - Timestamp

  For example: if *get request "/api/start"* is sent, parking meter app will return aforementioned JSON representation, because this should be a *post request*.