**Parking lot Assignment: (by XiaoDong, Wang)**

Design a parking lot management system that allows parking lot with N entries and M exits and control the entry and exit count. It should be able to identify the capacity and lock the entries until parking is available again.

The implementation should take into account

• General development principles / guidelines,

• Code building mechanism,

• Multithreading use cases,

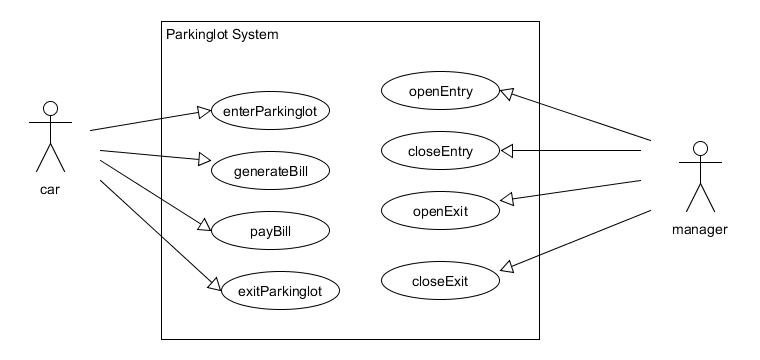
• Extensibility

• Maintainability

• Testability

• Future enhancement to be able to consume coins of different countries.

***Introduction of design***



The required parking lot has a fixed number of parking spaces. The total vehicles in the parking lot should not be greater than the number.

The parking lot has a fixed entries and exits. Each entry and exit can be set to open and closed. Vehicles are allowed to pass the open gates concurrently.

When a vehicle entered the parking lot, it got a parking token with time stamp.

The token can be used to get an unpaid parking bill. In this case, we assume each token is 10 dollars.

A parking bill can be paid by giving a collection of payable. A payable could be any currency of coin, check and paper money, etc., as long as they can be exchanged to dollar. In this case, currency exchange is not implemented.

A vehicle should provide a paid parking bill before going out of the parking lot.

***Multithreading use case:***

\* Description of the Multithreading use case

\*

\* It demonstrates concurrent access of resource, thread execution,

\* thread lock and result gathering.

\*

\* Prepare:

\*

\* A parking lot is constructed with 2 entries, 2 exits and 3 parking

\* spaces. Each entry or exit allows multiple cars passing at same time.

\* Once parking lot is full, cars are not allowed to enter

\*

\* A car behaves like this: Wait until a group signal is ready, enter

\* the parking lot, stay inside for a while, pay bill, leave and save

\* journey.

\*

\* Test case:

\*

\* Send the car group to the parking lot. Once a car

\* is back, print its journey and send it to the parking lot again.

\* Repeat this three times.

\*

\* Result:

\*

\* Running the case, We expect to see the 4 cars group went to the 3

\* spaces parking lot three times. In each time, they started at the

\* same time. There is always one car blocked. When coming back, first

\* car got its journey printed first.

\*