Parking Lot

Requirements

1) Able to park the vehicle
2) Able to park two types of vehicles:- Bike and car
3) Aware about number of available/ blocked slots.
4) Charge the vehicles based on the time they spent in the parking lot.

On any project, try to find out who are the actors or identify all the nouns in ~~vehicle~~ parking-lot-system

Designs, are never best, it is always iterative.

Nouns:-

1. Vehicle
2. Parking-area
3. Ticket
4. Parking-slot
5. Gate
6. User
7. Address
8. Payment System

once, nouns are finalized or ~~to~~ entities are finalized

We have to define relevant attributes of a vehicle

Vehicle:- (as an interface)

① Registration Number ③ State ④ Country
② Type → Two Wheeler / Four Wheeler ⑤ Parking Ticket

Note:- When we are designing a system, we are ~~only~~ not interested in ~~marking~~ knowing all the attributes of a given entity. We are only interested in those attributes which is minimum and needed for our functioning.

Address:-

↳ id
↳ Pincode
↳ street1, street2, city, country

Parking Area

↳ name
↳ Address, List of parking slot

Ticket

1) id
2) Time stamp
3) Type
   ↳ [Enum Car/Bike etc]

Parking slot

① id
② Type → Enum[Bike/Car]
③ isOccupied (Bool)
④

(methods)
To find number of ~~free~~ empty slots
number of filled slots

Gate:-

↳ id
↳ type (Enum: Entry / Exit)

↓
Additional responsibility of charging the vehicles.
↓
(Ticket, Payment System)

User:-

Name, Address,

↙ ↘

Customer
↳ vehicle
↳ CustomerID

Admin
vehicle, manage parking slot
parking slot plot

Pattern :- Creational Design Pattern

Why :- We are creating object of Vehicle, it can be Car, Auto, Bike, Scooti, Truck, Bus etc, User etc.

User (Single hona chahiye) → Singleton Design Pattern

Vehicle ( ) → Factory Design pattern.

Vehicle (interface)

① get_Registration_Number ( ) → String
② get_State ( ) → String
③ get_Country ( ) → String
④ get_Type ( ) → TwoWheeler / ThreeWheeler / FourWheeler
⑤ get_ParkingTicket ( ) → Return Object of ParkingTicket

There is no importance of parking ticket without the vehicle. So, we generate parking ticket for vehicle. We use builder design pattern. Once, we are defining vehicle object we instantiate parking ticket for it at the same time. We make use of nested class here.

Attributes of Vehicle Object :-

Instantiate :- Bike, Auto, Car ~~Scooter~~ ← Two/Three/Four Wheeler (Enum)

(a) RegistrationNo (b) State (c) Country (d) type (Enum)
(e) Parking_ticket (object) → Nested Class
↳ Use Parking_Ticket Builder

Address :-

① String id ② String Pincode } all are string
③ City ④ state ⑤ Country
⑥ ~~State~~ Place

↓ define interface

getstate() get_id()
get_place() get_pincode()
get_country() get_city()

Parking_Area

↓ interface

get_id() → return name
get_name()
get_address ( ) → return Address (Object)
~~get_ListofParkingSlot ( ) → display names~~
get_numberOfEmptySlot ( )
get_numberOfFilledSlot ( )

Attributes
(i) id
~~(ii) name~~
(iii) Address } String
~~(iv)~~

- Parking Ticket (Nested class)

① Ticket id
② Time stamp
} embeded inside vehicle so no vehicle type is required

interface → get_ticket_id()
get_time_stamp() → how many hours, vehicle is parked.

parking - slot

interface
- get_id
- get_type (Enum → Bike/Car/Auto)
- get_status (True/False is occupied)

Object – Attributes
(a) ID → string
(b) Type → Object of Vehicle
(c) Status → isOccupied (boolean)

Dropping idea of parking ticket nested class inside vehicle class as a large complexity is created, we can make parking ticket as extra class and parking ticket is associated with vehicle i.e it act as an attribute.

Entry Gate will have power to generate ticket

Now,

generateTicket (User ~~vehicle~~, ~~No~~ of hours)

↓ An an Argument → idea dropped.

Based on Type and vehicle registration number

required to park vehicle.

↳ Ticket is generated.

Idea of parking vehicle based on number of late hours is dropped.

Now, (vi) Make Exit Gate which have generateBill() function, take Ticket and number of hours vehicle parked as an argument

Now, (Parking Ticket Class Attribute)

(a) Ticket ID
(b) Vehicle Registration No.
(c) User name (d) User ID
(e) Charges/hour

Payment system

1) Take payment
   ↳ take ticket as argument and return the price in rupees/Dollars.

How do we stitch all pieces together, this is third step of low level design

(a) Parking Management system
   ↳ Separate file that stitch all pieces together

1. Initiate the parking place.
   ↳ create the slot objects
   ↳ create the parking area
   ↳ create the payment system
   ↳ create the gate objects.
2. Initialize ticket object
3. User object, vehicle object
4. Whenever the parking is done, update the flags.

User :-

Interface

① Customer ID → String
② Customer Name
③ Vehicle → ~~Object~~ List of Vehicle

get_CustomerID → return CustomerID
get_CustomerName → return CustomerName
get_Vehicle() → return Vehicle
④ display_category()

(There is possibility that user has more than one vehicle registered with his name)

Now, make use of Singleton design Pattern ~~by introducing User Account as Nested class in~~ while creating a user

Now, User is implemented as an Abstract Interface

Customer Interface
implements all user functionalities

Admin Interface
implements all user functionalities along with get_ParkingSlots()
Admin people who work in this system have vehicles too. So they are given their parking slot. So, these parking slots are reserved for them.

on display_category() we show either it is a customer or an employee function will be of void datatype.

Gate (Interface) (Abstract)

(i) get_id()
(ii) type (Entry/Exit) get_Type() → returns Enum
(iii) getParkingArea() → return ParkingArea

Entry (Interface)
implements gate interface with generation of ticket system
generate_Ticket()
↳ generate Vehicle

Exit (Interface)
implements gate interface with generation of the payment system
generate_Payment or generate_Bill (Vehicle)
↳ Take Vehicle as argument & return amount.

Note:- Each vehicle object you associate with Builder Design pattern, you must give charge price per hour

For Ex:- Bike → ₹105 for 1 hour

Provision of Late hours

Sometimes, a user may not come at time

So, Penalty of ₹ 1 for extra hours

For Car:- ₹ 10 for 1 hour

Penalty of ₹ 3 for Late hour

For Auto:- ₹ 8 for 1 hour

Penalty of ₹ 2 for Late hour.

SOLID → Design Principle (Maximum follow up)

Analysis:- Design Principle and Pattern Required:-

(a) Creational (All)

(i) Singleton

(ii) Factory

(iii) Builder (May be or not)

(b) Structural (May be very less used or not)

(i) Composite (Can be used)

(c) SOLID (Tried to follow in each component of project).

Implementation in Parking-Lot-App

(a) Steps to proceed:-

(i) Initialize parking slot ← dependency

(ii) Initialize Parking Area ← dependency

(iii) Gate (Entry) ← dependency

(iv) Initialize vehicle objects ← dependent on this

(v) Initialize Users

Generate Address for Parking Area (Mediator step)

Before initializing vehicle objects, Gate must be initialized, Gate must have parking area inside itself as an attribute. Because gate must belong to parking area.

// getters & setters in each class

Vehicle Renting System

Requirements [redacted]

(1) User Management → (Singleton Design Pattern)
   ↳ Distinct Roles: Admin & Customer

(2) Vehicle Management
   (a) Admin should be able to add, update, or remove vehicles
      ↳ Type of Fleet we need to make that will be collection of vehicles
      ↓
      Composite design pattern

   (b) Vehicle should be categorized
      ↙ Cars ↓ Bikes ↘ Trucks → Auto (etc.)
      [Factory design pattern, we can use here ←
      ↳ Object creation (Creational design Pattern)]

   (c) No need to use Admin. We took customer only in User Management. As admin is one who will handle this program based on user request

   (d) Vehicle attributes
      ID, type, ~~model~~, rental price and availability status (Available / Unavailable)

(3) Payment Processing.
   Based on number of days, there should be provision to generate Bill

(4) ~~Rental system~~
   Based on

Now, identifying the actors of Vehicle Renting system

(1) Vehicle
(2) User (Customer)
(3) EKart
(4) Payment_System
(5) Ticket

Assuming that a user can take multiple vehicles at same time. So, EKart is there.

(1) Vehicle (Interface)

Bike Car Truck Auto (etc)

get_ID, get_type, get_Rental price (no of days)
get_status (bool);
↳ bool (Return)
↓
True → Available
False → Unavailable.

on the basis of number of days, rent is charged.

Use of factory Design Pattern

(2) User (Customer)

(1) User ID
(2) User Name
(3) User PhoneNumber
(4) List of Vehicles

} String } → Interface

get_ID()
get_name()
get_phonenum()

↳ (3) EKart as an object → display_vehicle_Details
add_vehicle()
remove_Vehicle()

(4) Ticket Generate System

(1) User ID
(2) EKart
(3) Ticket ID
generateTicket(User)
Take user as input and return Ticket obj to App

(5) Payment System
↓ Interface
generate bill()
(a) int noOfDays;
(b) Ticket → object
Attributes

// Getters & setters in each class

Step by step :-

1. Initialize Vehicle Objects
2. Initialize Cost Object
3. Initialize User Object
4. Initialize Ticket Objects
5. Initialize Payment Objects