How to design a parking lot using object-oriented principles?

Design a parking lot using object-oriented principles.

Asked In : Amazon, Apple, Google and many more interviews

Solution: For our purposes right now, we'll make the following assumptions. We made these specific assumptions to add a bit of complexity to the problem without adding too much. If you made different assumptions, that's totally fine.

1) The parking lot has multiple levels. Each level has multiple rows of spots.

2) The parking lot can park motorcycles, cars, and buses.

3) The parking lot has motorcycle spots, compact spots, and large spots.

4) A motorcycle can park in any spot.

5) A car can park in either a single compact spot or a single large spot.

6) A bus can park in five large spots that are consecutive and within the same row. It cannot park in small spots.

In the below implementation, we have created an abstract class Vehicle, from which Car, Bus, and Motorcycle inherit. To handle the different parking spot sizes, we have just one class ParkingSpot which has a member variable indicating the size.

Main Logic in Java given below

```
// Vehicle and its inherited classes.
public enum VehicleSize { Motorcycle, Compact,Large }

public abstract class Vehicle
{
      protected ArrayList<ParkingSpot> parkingSpots =
                           new ArrayList<ParkingSpot>();
      protected String licensePlate;
      protected int spotsNeeded;
      protected VehicleSize size;

      public int getSpotsNeeded()
```

```
        {
            return spotsNeeded;
        }
        public VehicleSize getSize()
        {
            return size;
        }

        /* Park vehicle in this spot (among others,
           potentially) */
        public void parkinSpot(ParkingSpot s)
        {
            parkingSpots.add(s);
        }


        /* Remove vehicle from spot, and notify spot
           that it's gone */
        public void clearSpots() { ... }

        /* Checks if the spot is big enough for the
           vehicle (and is available).
           This * compares the SIZE only.It does not
          check if it has enough spots. */
        public abstract boolean canFitinSpot(ParkingSpot spot);
}

public class Bus extends Vehicle
{
    public Bus()
    {
        spotsNeeded = 5;
        size = VehicleSize.Large;
    }

    /* Checks if the spot is a Large. Doesn't check
     num of spots */
    public boolean canFitinSpot(ParkingSpot spot)
    {... }
}

public class Car extends Vehicle
{
    public Car()
    {
        spotsNeeded = 1;
        size = VehicleSize.Compact;
    }

    /* Checks if the spot is a Compact or a Large. */
    public boolean canFitinSpot(ParkingSpot spot)
    { ... }
}

public class Motorcycle extends Vehicle
{
    public Motorcycle()
    {
        spotsNeeded = 1;
        size = VehicleSize.Motorcycle;
    }
    public boolean canFitinSpot(ParkingSpot spot)
    { ... }
}
```

The **ParkingSpot** is implemented by having just a variable which represents the size of the spot. We could have implemented this by having classes for LargeSpot, CompactSpot, and MotorcycleSpot which inherit from ParkingSpot, but this is probably overkilled. The spots probably do not have different behaviors, other than their sizes.

```
public class ParkingSpot
{
    private Vehicle vehicle;
    private VehicleSize spotSize;
    private int row;
    private int spotNumber;
    private Level level;

    public ParkingSpot(Level lvl, int r, int n,
                       VehicleSize s)
    { ... }

    public boolean isAvailable()
    {
        return vehicle == null;
    }

    /* Check if the spot is big enough and is available */
    public boolean canFitVehicle(Vehicle vehicle) { ... }

    /* Park vehicle in this spot. */
    public boolean park(Vehicle v) {..}

    public int getRow()
    {
        return row;
    }
    public int getSpotNumber()
    {
        return spotNumber;
    }

    /* Remove vehicle from spot, and notify
      level that a new spot is available */
    public void removeVehicle() { ... }
}
```

Run on IDE

Source :

www.andiamogo.com/S-OOD.pdf

More References :

http://qa.geeksforgeeks.org/548/how-to-design-a-parking-lot

https://www.quora.com/How-do-I-answer-design-related-questions-like-design-a-parking-lot-in-an-Amazon-interview

http://stackoverflow.com/questions/764933/amazon-interview-question-design-an-oo-parking-lot

This article is contributed by **Mr. Somesh Awasthi**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your a... to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page a... help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Corner Company Wise Coding Practice

Design Pattern

Recommended Posts:

Design data structures and algorithms for in-memory file system
Design Patterns | Set 1 (Introduction)
How to prevent Singleton Pattern from Reflection, Serialization and Cloning?
Singleton Design Pattern | Implementation
Design Patterns | Set 2 (Factory Method)
Data Access Object Pattern
Front Controller Design Pattern
Business Delegate Pattern
MVC Design Pattern
Intercepting Filter Pattern