Explain the design pattern used in following:

interface Vehicle {

int set\_num\_of\_wheels()

int set\_num\_of\_passengers()

boolean has\_gas()

}

a) Explain how can you use the pattern to create car and plane class?

**Ans:** First of all, set function should not return any value.

In addition, from only one interface it is difficult to understand its design pattern. I assume this is factory design pattern. We will keep Vehicle class as a base class and car, plane will be subclasses of Vehicle class.

The key point of factory design pattern we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

Here the subclasses are Car and Plane class

For Example-

Class Car implemnts Vehicle{

public int set\_num\_of\_wheels(){

return 4;

}

public int set\_num\_of\_passengers(){

return 4;

}

public boolean has\_gas(){

return true;

}

}

Class Plane implements Vehicle{

public int set\_num\_of\_wheels(){

return 3;

}

public int set\_num\_of\_passengers(){

return 150;

}

public boolean has\_gas(){

return false;

}

}

**We should create a factory class to create different objects**.  
  
public class VehicleFactory {

public Vehicle getType(String vehicleType){

if(vehicleType == null){

return null;

}

if(vehicleType.equalsIgnoreCase("CAR")){

return new Car();

} else if(vehicleType.equalsIgnoreCase("PLANE")){

return new Plane();

}

return null;

}

}

**#Driver Class**

public class FactoryPatternDemo {

public static void main(String[] args) {

VehicleFactory vehicleFactory = new VehicleFactory ();

Vehicle car = vehicleFactory.getType("CAR");

Vehicle plane = vehicleFactory.getType("PLANE");

}

}

b) Use a different design pattern for this solution.

I can use Decorator design pattern for this solution.

Decorator design pattern is used to modify the functionality of an object at runtime, allows to add new functionality and other instances of the same class will not be affected by this, so individual object gets the modified behavior. Decorator design pattern is one kind of the structural design pattern.

**# Solve with Decorator design pattern**

interface Vehicle {

int set\_num\_of\_wheels()

int set\_num\_of\_passengers()

boolean has\_gas()

}

public class MainVehicle implements Vehicle {

protected Vehicle vehicle;

public MainVehicle(Vehicle c){

this.vehicle=c;

}

@Override

public int set\_num\_of\_wheels() {

this.vehicle.set\_num\_of\_wheels();

}

@Override

public int set\_num\_of\_passengers() {

this.vehicle.set\_num\_of\_passengers();

}

@Override

public boolean has\_gas() {

this.vehicle.has\_gas();

}

}

// **concrete decorator classes**

public class Car extends MainVehicle {

public Car(Car c) {

super(c);

}

@Override

public int set\_num\_of\_wheels(){

super.set\_num\_of\_wheels();

return 4;

}

@Override

public int set\_num\_of\_passengers(){

super.set\_num\_of\_passengers();

return 5;

}

@Override

public boolean has\_gas(){

super.has\_gas();

return true;

}

}

public class Plane extends MainVehicle {

public Plane(Plane p) {

super(p);

}

@Override

public int set\_num\_of\_wheels(){

super.set\_num\_of\_wheels();

return 3;

}

@Override

public int set\_num\_of\_passengers(){

super.set\_num\_of\_passengers();

return 150;

}

@Override

public boolean has\_gas(){

super.has\_gas();

return false;

}

}