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

Answer:

The given interface is,

```
public interface Vehicle {
   int set_num_of_wheels();
   int set_num_of_passengers();
   boolean has_gas();
}
```

The other two classes that are given [Car and Plane]. Both of these classes can implement that interface in order to avail all the functionalities that lies in Vehicle interface.

And the implementation should like in the following way,

1.1 Car class:

```
public class Car implements Vehicle{
   @Override
   public int set_num_of_wheels() {
      return 0;
   }
   @Override
   public int set_num_of_passengers() {
      return 0;
   }
   @Override
   public boolean has_gas() {
      return true;
   }
}
```

1.2 Vehicle class:

```
public class Plane implements Vehicle{
    @Override
    public int set_num_of_wheels() {
        return 0;
    }
    @Override
    public int set_num_of_passengers() {
        return 0;
    }
    @Override
    public boolean has_gas() {
        return true;
    }
}
```

Both the class Car and Plane implements the Vehicle interface because both of the class has those common properties which they shared between each other.

Question b) Use a different design pattern for this solution.

Answer:

There are two classes and they are Car and Plane. As they have common like passengers, wheels and gas. So in a word they will share a single class which will have those common properties. In here we can use **Factory Design Pattern**. Although **Delegate Design Pattern** can be used here. There will be a factory class which will provide the instance based on the requirement. Let's see the implementation,

The Vehicle class will look like,

```
public class Vehicle {
   int set_num_of_wheels(){
      return 0;
   }
   int set_num_of_passengers(){
      return 0;
   }
   boolean has_gas(){
      return true;
   }
}
```

The factory class will look like following,

```
public class GetVehicle {
   public Vehicle getVehicle(String vehicleType){
      if (vehicleType.equals("Car")) {
         return new Car();
      }
      else if (vehicleType.equals("Plane")) {
         return new Plane();
      }
      return null;
   }
}
```

The car and plane class will extend this Vehicle class and its properties.

1.1 Car class:

```
public class Car extends Vehicle {
    @Override
    public int set_num_of_wheels() {
        return 0;
    }
    @Override
    public int set_num_of_passengers() {
        return 0;
    }
    @Override
    public boolean has_gas() {
        return true;
    }
}
```

1.2 Plane class:

```
public class Plane extends Vehicle {
    @Override
    public int set_num_of_wheels() {
        return 0;
    }
    @Override
    public int set_num_of_passengers() {
        return 0;
    }
    @Override
    public boolean has_gas() {
        return true;
    }
}
```