Analysis

1) Encapsulate what varies

In the run part of the car class we have written different methods to fit different parts like fitEngine() , fitChasis() etc.
what we could've done is make a single method fitPart() and pass the part to be fitted as an argument to it.

2) Favour composition over inheritance

We have the favoured composition over inheritance as it is evident from our code in Car class. Using this approach, we have composed the car class using different parts. It gives us the flexibility to add more parts (if required) very easily. We just have to add an attribute if we are required to change the code to add more parts.

3) Strive for loose coupling between objects that interact

In our code, we have strived for loose coupling between different parts and tried that these share least dependency on each other. The parts interact in the sense that each part in a particular assembly line is fitted to the same car. The best thing about this is that, one part being fitted does not affect the other in any way.

4) Depend on abstraction, do not depend on concrete classes

We have not used this design pattern. All of the classes in our program are concrete. We could have created the interfaces named "Parts" and implemented it in all the individual concrete classes for parts. In that way, our code would be more readable and would provide us with more elasticity when we define more specific classes.