Analysis of OOP Principes and Design Pattern

Strategy Design Pattern and **Factory Design Pattern** is followed by code written by us.

Strategy Design Pattern :

In our code, we have made Car Interface which has methods like frame(), wheel(), engine(), interior(), chasis(), color(). We realized later that even if we had defined Frame f; Wheel W; Engine e; Interior it; Chasis ch; Color c; inside car interface then that would also be fine as we can make object of our required class later in implementation using new Keyword.

Also we made different interfaces for all basic parts as there can be variety in each basic part. Then we made concrete classes for all varieties of basic parts which implements interface of respective basic part .

(Example: Frame may be available of in market of different company and we named those as frame1, frame2, and so on…)

Then we made AssemblyLine Class which implements Car interface and define all methods present in Car Interface.

Factory Design Pattern :

A factory Design Pattern is followed in our code as irrespective of what type of car you make (i.e. any kind of object) our code will not change which agree with factory Design Statement, that it can handle object creation and encapsulates it in a subclass because of which it decouples the client code in the superclass from the object creation code in the subclass.

Future Applications :

In future even if more varieties of basic parts comes in market then also our

Car interface and interface of each basic components will remain same. We just need to add concrete classes which will implement it's respective basic part interface. (which is very good point even if in future let's say 10 more varieties of frame comes we only need to add concrete class for each new variety, rest code will remain same.)

1) **Encapsulate what varies** :

   This Principle is followed as in our code we have made separate interfaces of each basic parts just because each basic part may **vary** from car to car. Also we made a Car interface.

2) **Favour composition over Inheritanc**e :

   In our code we have used inheritance only to extend Thread class, in rest of our code we have made interfaces and implement those interfaces in concrete classes. Also as for car all varieties of basic parts may vary from car to car so in our code we were able to use interfaces more.

3) **Program to an interface not implementatio**n :

   In our code we have implemented our best so that whichever car we want to make can be made at run-time whatever is input given by user. Also we have made many interface and implement those interfaces in concrete classes which in turn gives us flexibility to make any type of car at run-time.

4) **Classes should be open for extension and closed for modification** :

   As we discussed earlier in Future Applications even if more varieties of basic parts comes in market then too there will be **no modification** in

   Car Interface and interfaces of each basic part. We just need to add more concrete classes as per varieties coming in market.(**Just Extension**)