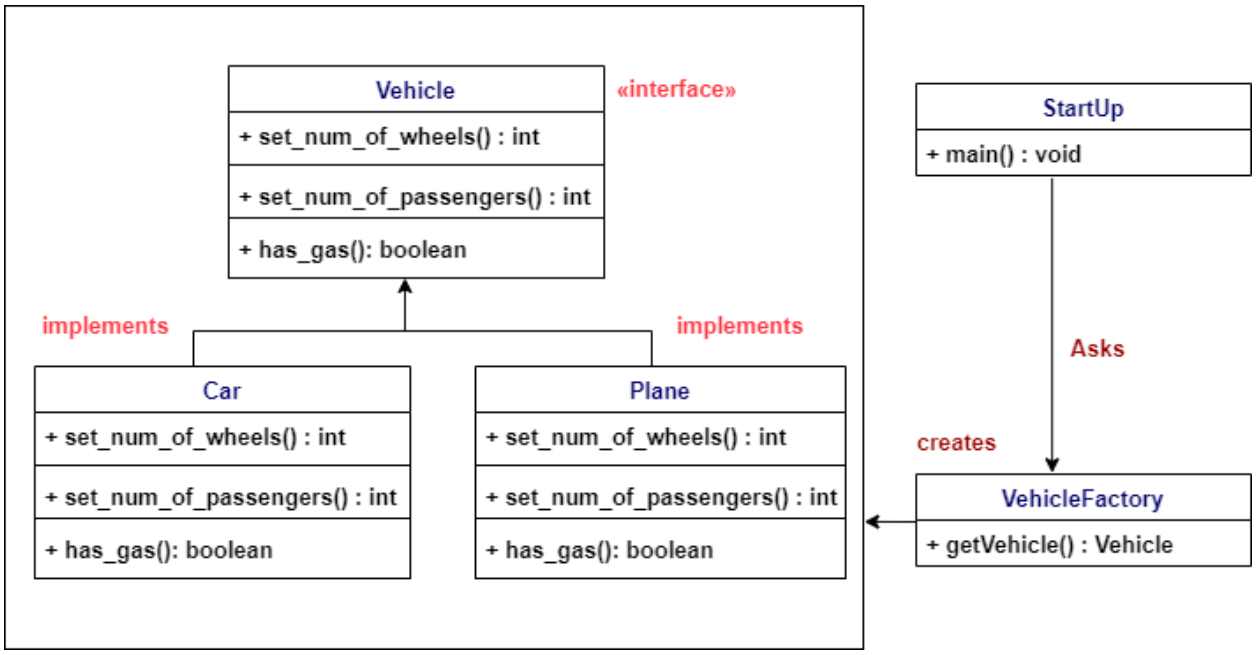
Design Pattern

Design patterns represent the best practices used by experienced object-oriented software developers. Design patterns are solutions to general problems that software developers faced during software development. These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time. We must use the design patterns **during the analysis and requirement phase of SDLC**(Software Development Life Cycle).Design patterns ease the analysis and requirement phase of SDLC by providing information based on prior hands-on experiences.

Here we have an interface **Vehicle**, which will be impairments by Car and Plane class. Now I will be explaining how we can use the different pattern to create car and plane class bellow to implements the scenario:

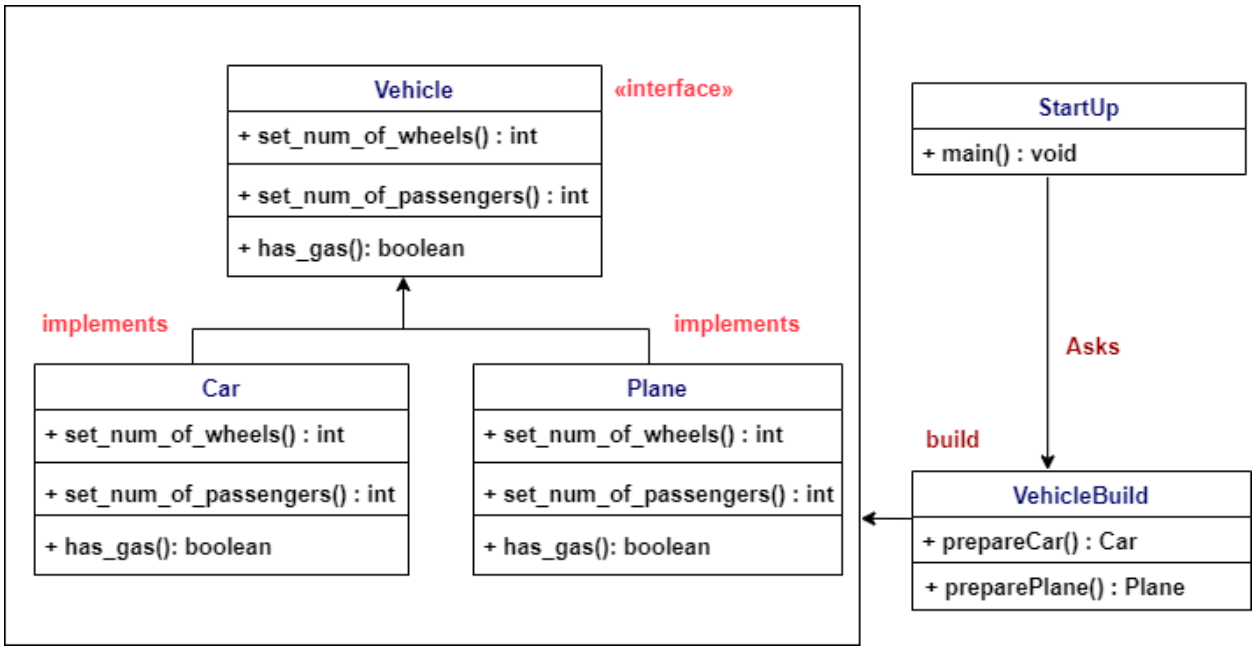
i. Factory Pattern

This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object. In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface. Such as,



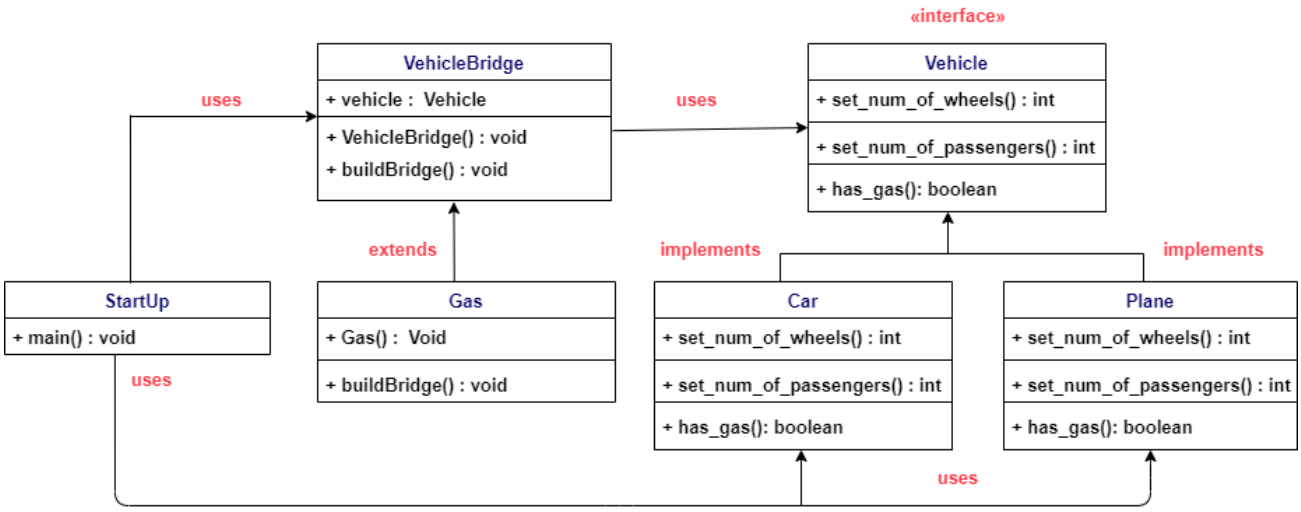
ii. Builder Pattern

A Builder class builds the final object step by step. This builder is independent of other objects. For example of our prototype about builder pattern:



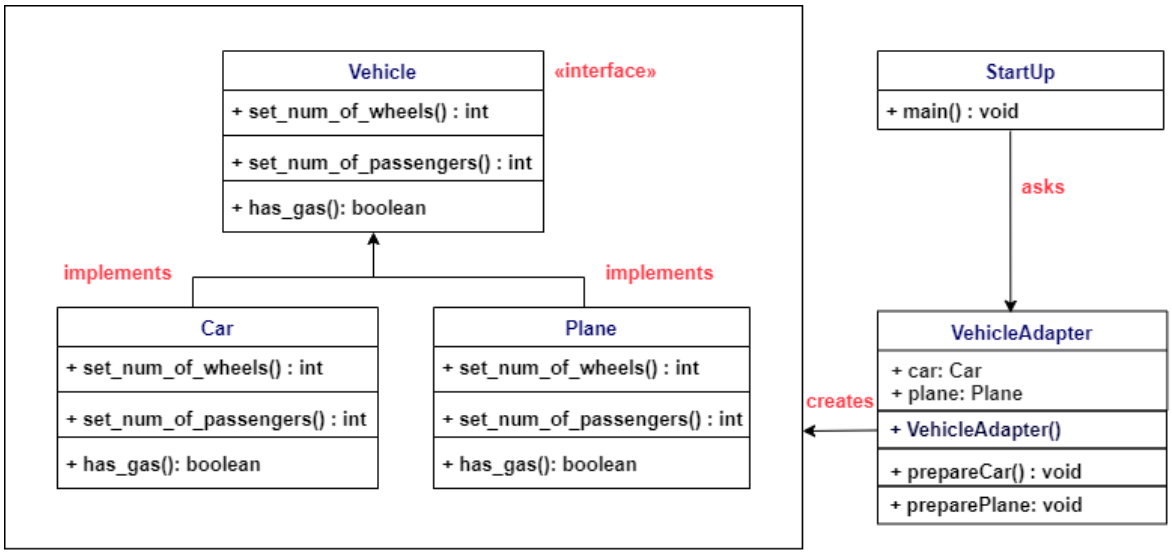
iii. Bridge Pattern

Bridge is used when we need to decouple an abstraction from its implementation so that the two can vary independently. Here is the example of bridge pattern for our prototype.



iv. Facade Pattern

Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under structural pattern as this pattern adds an interface to existing system to hide its complexities. Such as,



v. Proxy Pattern

In proxy pattern, a class represents functionality of another class. This type of design pattern comes under structural pattern. According to our prototype its will be,