#Question no 2 of Bongo's Android Developer written test.

#Problem
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 you can use the pattern to create car and plane class?
b) Use a different design pattern for this solution.

#Answer to the question number 2(a)

#Write Car and Plane class and use Vehicle interface.

1.Creat Vehicle interface under InerfaceCarAndPlane class.

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

2.Creat Car Class and implement Vehicle interface.
*/

```
package inerface.car.and.plane;


public class CarClass implements Vehicle{

      @Override
      public int set_num_of_wheels() {
                        return 4;

      }

      @Override
      public int set_num_of_passengers() {
               return 10;
      }

      @Override
      public boolean has_gas() {
            return true;
      }

}
```

/*

3.Creat Plane Class and implement Vehicle interface.

*/

```
package inerface.car.and.plane;

public class PlaneClass implements    Vehicle{

    @Override
    public int set_num_of_wheels() {

        return 8;
    }

    @Override
    public int set_num_of_passengers() {

        return 200;
    }

    @Override
    public boolean has_gas() {

        return true;
    }

}
```

*/

4.Creat an Object of Car and Plane class then print all function return value under InerfaceCarAndPlane class .

```
public class InerfaceCarAndPlane{


    public static void main(String[] args) {
        // TODO code application logic here
        CarClass carClass=new CarClass();
        PlaneClass planeClass=new PlaneClass();

        System.out.println("CarClass Method Return Data which i use Vehicale Interface");
        System.out.println(carClass.set_num_of_wheels());
          System.out.println(carClass.set_num_of_passengers());
          System.out.println(carClass.has_gas());
          System.out.println("PlaneClass Method Return Data which i use Vehicale Interface");
          System.out.println(planeClass.set_num_of_wheels());
          System.out.println(planeClass.set_num_of_passengers());
          System.out.println(planeClass.has_gas());



    }


}
```

#Unit test Of function.

```
// Test of main method, of class InerfaceCarAndPlane.

@Test
public void testMain() {
        System.out.println("main");
        String[] args = null;
        InerfaceCarAndPlane.main(args);
        // TODO review the generated test code and remove the default call to fail.
        fail("The test case is a prototype.");
}


   //Test of set_num_of_wheels method, of class Vehicle.

@Test
public void testSet_num_of_wheels() {
        System.out.println("set_num_of_wheels");
        Vehicle instance = new VehicleImpl();
        int expResult = 0;
        int result = instance.set_num_of_wheels();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        fail("The test case is a prototype.");
}


  // Test of set_num_of_passengers method, of class Vehicle.

@Test
public void testSet_num_of_passengers() {
        System.out.println("set_num_of_passengers");
        Vehicle instance = new VehicleImpl();
        int expResult = 0;
        int result = instance.set_num_of_passengers();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        fail("The test case is a prototype.");
}


   //Test of has_gas method, of class Vehicle.

@Test
public void testHas_gas() {
        System.out.println("has_gas");
        Vehicle instance = new VehicleImpl();
        boolean expResult = false;
        boolean result = instance.has_gas();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        fail("The test case is a prototype.");
}
```