**Chapter 1**

SimpleUDuck App.

We need to develop an app to simulate ducks, this app will be changed by time.

1. At the first app would be very simple, we have a Duck and two types of ducks.
2. Duck methods [quack , swim and display]
3. 2 sub types of ducks [MallrodDuck and RedHeadDuck].
4. Each type will implement the display method.

Problem no 1 :- we need to add fly functionality to our app.

Solution no 1 :- add fly() to the Duck superclass. And implement this method to be reused across all duck types.

Solution issue :- Some of ducks does not fly and this is wrong behaviour.

Solution no 2 :- don’t implement the fly method in superclass , just add the method declaration , then implement the fly method in the sub-classes.

Solution no 2 issue , some duck dose not fly and you will override the method to do anything.

Solution No 3 , replace the superclass with interface , then create an interface Flyable.

Then if duck can fly then it will implement this interface along with the duck interface.

Solution No 3 issue :- code will be duplicated , when we used the super-class , we were able to implement the fly behaviour in the base class.

No we can’t do the same in the interface, as we can’t use declaration only no implementation and thus we have to implement the fly method in all the sub-classes.

In this chapter there are 3 Design principles

1. Identify the aspects of your application that vary and separate them from what stays the same.
2. Program to an interface not implementation.
3. Favor composition over inheritance.

One constant in the software development **“CHANGE” .**

The solution for the duck problem.

1. According to first design principle , the duck behaviour will live in separate class.
2. Create FlyBehaviour interface and create classes need to implement it like FlyWithWings class and FlyNoWay Class.
3. According to 2nd design principle , we should code to interface.

**Code to interface :-**

Duck duck = new MallrodDuck ().

**Code to implementation :-**

MallrodDuck duck = new MallrodDuck();

But what If we need to change the Flying behaviour in the runtime ??

1. In the Duck super-class create a behaviour for the Flying Behaviour Interface

Class Duck{

IFlyingBehaviour FlyingBehaviour;

}

1. Create new method that sets the flying behaviour dynamic

SetFlyingBehaviour(IFlayingBehaviour fb){

This. FlyingBehaviour = fb.

}

1. In no 4 and 5 we fixed the issue we mentioned at no 3 [change the Flying behaviour in the runtime]