**Programing to an Interface**

I think it will better to discuss about design principals or you can say design tool based on which design patter are created.

The very first principal is ***“Programming to an Interface”*** what does it mean***?***

**Let try to understand the principal in details.**

In general scenario if you look at any problem statement or business solution you can find two part

1. Fixed part.
2. Variable

Fixed part is some kind boiler plate code. But when we design we take care of the variable part.

All the design patterns are discovered to maintains this variable parts. Because they are ever-changing. If your code is not flexible with future enhancement, then your code is not up to the mark.

***The question is in java How you can maintain your variable parts?***

To drill down to answer let magnify the statement maintain variable parts. We can think variables parts mean we know it will perform an operation but we don’t know what the operation is specifically we know the type of the operation but don’t know details of the implementation moreover client needs can be changed so that can be change in future.

Let’s, take an Example A computer monitor is design for display purpose. So I can say if Computer is a product and Computer monitor is a part of computer product. Then Computer monitor is responsible for **display operation.**

**Now later on client needs is changed now they wants to display on Projector**

So If our solution is not capable of welcome this needs it will be nothing but a waste product.

According to new needs what we can analyze is it will perform same action display but the module should change from Computer monitor to Projector.

So display module in Computer product should be flexible so we can change it easily. Or we can change it dynamically(runtime). We can say display module is like a strategy and client change the strategy now.

So our java solution is like following.

**interface displayModule**

**{**

**public void display();**

**}**

**Public class Monitor implements displayModule**

**{**

**public void display()**

**{**

**s.o.p(“Display through Monitor”);**

**}**

**Public class Projector implements displayModule**

**{**

**public void display()**

**{**

**s.o.p(“Display through projector”);**

**}**

**Public class Computer**

**{**

**displayModule dm;**

**public void setDisplayModule(displayModule dm)**

**{**

**this.dm=dm;**

**}**

**Public void display()**

**{**

**dm.display();**

**}**

**Public static void main(String args[])**

**{**

**Computer cm =new Computer();**

**displayModule dm = new Monitor();**

**displayModule dm1 = new Projector();**

**cm. setDisplayModule(dm);**

**cm. display();**

**cm. setDisplayModule(dm1);**

**cm. display();**

**}**

**}**

Look the solution we know Display Module should be flexible and we know the operation is display but according to client it may be change later but in computer there should be a display but we don’t know what will that equipment it’s may be monitor or projector or any other.

So we create an interface and every Display parts should implement that and provide the own definition of display.

Look the computer class here I create has-A relation call displaymodule because we know display module change frequently as per client need so always make as abstract

So we can change it runtime with actual implementation.

**Remember always code through an interface so you can change your strategy runtime with actual implementation.**

**Interface means here java interface or abstract class.**

**Make the variable parts as interface or abstract class as you know the operation which is never changes but it implementation or the implement module can change.**