Design Thoughts ON Open/Close principal

According to Bertrand Meyer

***“software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification***”

Read it the first time one can think the definition is a bit abstract as design principals are.

But the aforesaid statement is very powerful and can save you from fragile design and wired bugs.

The statement says that Design should be done such a way so it can welcome new changes

but without tampering the old implementation. Because your old implementation is tested and run fine in production, to welcome new changes it is not good to break old ones. So this principal says welcome new changes on top of old changes.

Let take an example, suppose requirement is to create a class which determines the area of a Rectangle.

This is very easy to develop for an OO Developer

Initial code like

**package** com.example.openclose;

**public** **class** AreaBuilder {

**public** **void** calculateRectangleArea(**int** length,**int** breadth)

       {

              System.***out***.println("Area of Rectangle is " + length\*breadth);

       }

**public** **static** **void** main(String[] args) {

              AreaBuilder builder = **new** AreaBuilder();

              builder.calculateRectangleArea(20, 10);

       }

}

The code is tested and then deployed.  But after some day new feature request raise now this class should able to compute the area of the square also.

Inexperienced developer will incorporate the requirements in following way

**package** com.example.openclose;

**public** **class** AreaBuilder {

**private** **void** calculateRectangleArea(**int** length,**int** breadth)

       {

              System.***out***.println("Area of Rectangle is " + length\*breadth);

       }

**private** **void** calculateSquareArea(**int** length)

       {

              System.***out***.println("Area of Square is " + length\*length);

       }

**public** **void** getArea(String type,**int** length,**int** breadth)

       {

**if**("square".equalsIgnoreCase(type))

              {

                     calculateSquareArea(length);

              }

**else** **if**("rectangle".equalsIgnoreCase(type))

              {

                     calculateRectangleArea(length,breadth);

              }

       }

**public** **void** getArea(String type,**int** length)

       {

              getArea(type,length,length);

       }

**public** **static** **void** main(String[] args) {

              AreaBuilder builder = **new** AreaBuilder();

              //builder.calculateRectangleArea(20, 10);

              builder.getArea("square", 10);

              builder.getArea("rectangle", 10,5);

       }

}

Certainly, it will meet the requirements but break the Open close principal. Also, this code has some shortcomings

1.    To incorporate new changes, we will modify the old class so it breaks the statement “closed for modification”.

2.    The new code can break old functionality if the implementation is wrong, so we not only fail to deliver new code but also break the old functionality which is not intended.

3.    For every new shape added in the system, we need to modify this class and make the class more complex and less cohesive.

To overcome aforesaid problems, we try to follow the principal refactor the code.

*“Open for extension closed for modification”*

In java, the extension can achieve through inheritance. So we can create an interface called “shape” and a method called area() in it so every shape can implement area method and provide its implementation.

So if a new shape is coming we just create a new class and implement shape interface if anything goes wrong only that shape class will be impacted not the previous implementations thats why the Open close principal is so important in design.

Refactor code:

**package** com.example.openclose;

**public** **interface** shape {

**public** **void** area();

}

**package** com.example.openclose;

**public** **class** Rectangle **implements** shape{

**private** **int** length;

**private** **int** breadth;

       Rectangle(**int** length,**int** breadth)

       {

**this**.length=length;

**this**.breadth=breadth;

       }

       @Override

**public** **void** area() {

              System.***out***.println("Area of Rectangle is " + length\*breadth);

       }

}

**package** com.example.openclose;

**public** **class** Square **implements** shape{

**private** **int** edge;

**public** Square(**int** edge)

       {

**this**.edge=edge;

       }

       @Override

**public** **void** area() {

              System.***out***.println("Area of Square is " + edge\*edge);

       }

}

**package** com.example.openclose;

**public** **class** ShapeManager {

**private** **static** ShapeManager *manager* = **new** ShapeManager();

       shape shape;

**private** ShapeManager()

       {

       }

**public** **static** ShapeManager getInstance()

       {

**return** *manager*;

       }

**public** **void** getArea() **throws** IllegalArgumentException

       {

**if**(shape ==**null**)

              {

**throw** **new** IllegalArgumentException("please provide a Shape");

              }

              shape.area();

       }

**public** shape getShape() {

**return** shape;

       }

**public** **void** setShape(shape shape) {

**this**.shape = shape;

       }

**public** **static** **void** main(String[] args) {

              shape rect = **new** Rectangle(10,20);

              shape square = **new** Square(10);

              ShapeManager instance = ShapeManager.*getInstance*();

              instance.setShape(rect);

              instance.getArea();

              instance.setShape(square);

              instance.getArea();

       }

}

Output :

 Area of Rectangle is 200

Area of Square is 100

Hooray, we refactor the code and it is maintaining the principal. We can incorporate any number of shape by adding new classes and without modifying the old classes.

***The client is happy now they want to introduce a brand new feature Now shape can be drawn on a paper.***

So inexperinced developer thought it is easy task to do just add a new method call drawShape() on shape interface it will meet client needs and not breaking the Open Close principle

Let’s do that,

**package** com.example.openclose;

**public** **interface** shape {

**public** **void** area();

**public** **void** drawShape();

}

**package** com.example.openclose;

**public** **class** Rectangle **implements** shape{

**private** **int** length;

**private** **int** breadth;

       Rectangle(**int** length,**int** breadth)

       {

**this**.length=length;

**this**.breadth=breadth;

       }

       @Override

**public** **void** area() {

              System.***out***.println("Area of Rectangle is " + length\*breadth);

       }

       @Override

**public** **void** drawShape() {

              System.***out***.println("drawing recangle on paper" );

       }

}

**package** com.example.openclose;

**public** **class** Square **implements** shape{

**private** **int** edge;

**public** Square(**int** edge)

       {

**this**.edge=edge;

       }

       @Override

**public** **void** area() {

              System.***out***.println("Area of Square is " + edge\*edge);

       }

       @Override

**public** **void** drawShape() {

              System.***out***.println("drawing square on paper" );

       }

}

**package** com.example.openclose;

**public** **class** ShapeManager {

**private** **static** ShapeManager *manager* = **new** ShapeManager();

       shape shape;

**private** ShapeManager()

       {

       }

**public** **static** ShapeManager getInstance()

       {

**return** *manager*;

       }

**public** **void** getArea() **throws** IllegalArgumentException

       {

**if**(shape ==**null**)

              {

**throw** **new** IllegalArgumentException("please provide a Shape");

              }

              shape.area();

       }

**public** **void** draw() **throws** IllegalArgumentException

       {

**if**(shape ==**null**)

              {

**throw** **new** IllegalArgumentException("please provide a Shape");

              }

              shape.drawShape();

       }

**public** shape getShape() {

**return** shape;

       }

**public** **void** setShape(shape shape) {

**this**.shape = shape;

       }

**public** **static** **void** main(String[] args) {

              shape rect = **new** Rectangle(10,20);

              shape square = **new** Square(10);

              ShapeManager instance = ShapeManager.*getInstance*();

              instance.setShape(rect);

              instance.getArea();

              instance.draw();

              instance.setShape(square);

              instance.getArea();

              instance.draw();

       }

}

Output :

Area of Rectangle is 200

drawing recangle on paper

Area of Square is 100

drawing square on paper

So far so good. But is it really maintains Open/close principle? In bare eyes, yes but if we put our thinking cap, there is clue in client needs which can break this principal

**Clients want shape can be drawn on a paper but it can be drawn on Computer or in canvas also.**

So in future, if clients want to draw on the computer. To incorporate this change Open close principal will be broken. As again we need to put if /else check on each shape drawshape() method.

Now we will discuss How judiciously we can design open close principle?

1.    I classified Objects properties in two ways a. primary properties b. Composition properties.

2.    If any functionality(method) depends on only primary properties i.e which are the sole properties of that domain object.  You can declare them in the interface or abstract class from which this class inherited.

3.    If a functionality depends on an external entity always use composition rather than inheritance. (Here draw functionality depends on paper an external entity so we should use composition rather than inheritance)

Now refactor the code again to support draw function.

**package** com.example.openclose;

**public** **interface** shape {

**public** **void** area();

**public** **void** drawShape();

}

**package** com.example.openclose;

**public** **interface** drawAPI {

**public** String draw();

}

**package** com.example.openclose;

**public** **class** PaperDraw **implements** drawAPI{

       @Override

**public** String draw() {

**return** "paper";

       }

}

**package** com.example.openclose;

**public** **class** ComputerDraw **implements** drawAPI{

       @Override

**public** String draw() {

              // **TODO** Auto-generated method stub

**return** "computer";

       }

}

**package** com.example.openclose;

**public** **class** Rectangle **implements** shape{

**private** **int** length;

**private** **int** breadth;

       drawAPI api;

       Rectangle(**int** length,**int** breadth,drawAPI api)

       {

**this**.length=length;

**this**.breadth=breadth;

**this**.api=api;

       }

       @Override

**public** **void** area() {

              System.***out***.println("Area of Rectangle is " + length\*breadth);

       }

       @Override

**public** **void** drawShape() {

              String medium = api.draw();

              System.***out***.println("drawing recangle on " + medium);

       }

}

**package** com.example.openclose;

**public** **class** Square **implements** shape{

**private** **int** edge;

       drawAPI api;

**public** Square(**int** edge,drawAPI api)

       {

**this**.edge=edge;

**this**.api=api;

       }

       @Override

**public** **void** area() {

              System.***out***.println("Area of Square is " + edge\*edge);

       }

       @Override

**public** **void** drawShape() {

              String medium=api.draw();

              System.***out***.println("drawing square on "+medium );

       }

}

**package** com.example.openclose;

**public** **class** ShapeManager {

**private** **static** ShapeManager *manager* = **new** ShapeManager();

       shape shape;

**private** ShapeManager()

       {

       }

**public** **static** ShapeManager getInstance()

       {

**return** *manager*;

       }

**public** **void** getArea() **throws** IllegalArgumentException

       {

**if**(shape ==**null**)

              {

**throw** **new** IllegalArgumentException("please provide a Shape");

              }

              shape.area();

       }

**public** **void** draw() **throws** IllegalArgumentException

       {

**if**(shape ==**null**)

              {

**throw** **new** IllegalArgumentException("please provide a Shape");

              }

              shape.drawShape();

       }

**public** shape getShape() {

**return** shape;

       }

**public** **void** setShape(shape shape) {

**this**.shape = shape;

       }

**public** **static** **void** main(String[] args) {

              shape rect = **new** Rectangle(10,20,**new** ComputerDraw());

              shape square = **new** Square(10,**new** PaperDraw());

              ShapeManager instance = ShapeManager.*getInstance*();

              instance.setShape(rect);

              instance.getArea();

              instance.draw();

              instance.setShape(square);

              instance.getArea();

              instance.draw();

       }

}

Output :

Area of Rectangle is 200

drawing rectangle on computer

Area of Square is 100

drawing square on paper

Now draw() method is in shape class as we assume every shape is drawable and we create  composition between  shape family and drawAPI family.

But to maintain Open close principal we need to create a lot of classes. Make code complex.

**I throw an open question,**

 If a client is reluctant for change or development time is short or for a system where change requests are very less, then should we always maintain open/close principle or break that rule and make the code much simpler?