

Open Close Principle

A clever application design and the code writing part should take care of the frequent changes that are done during the development and the maintaining phase of an application...

NEW Forum Discussions: NEW

What Design Pattern Should I Choose?

Design Principles and Patterns

Open Close Principle

Motivation

A clever application design and the code writing part should take care of the frequent changes that are done during the development and the maintaining phase of an application. Usually, many changes are involved when a new functionality is added to an application. Those changes in the existing code should be minimized, since it's assumed that the existing code is already unit tested and changes in already written code might affect the existing functionality in an unwanted manner.

The Open Close Principle states that the design and writing of the code should be done in a way that new functionality should be added with minimum changes in the existing code. The design should be done in a way to allow the adding of new functionality as new classes, keeping as much as possible existing code unchanged. Intent

Software entities like classes, modules and functions should be open for extension but closed for modifications. Example

Bellow is an example which violates the Open Close Principle. It implements a graphic editor which handles the drawing of different shapes. It's obviously that it does not follow the Open Close Principle since the GraphicEditor class has to be modified for every new shape class that has to be added. There are several disadvantages:

- for each new shape added the unit testing of the GraphicEditor should be redone.
- when a new type of shape is added the time for adding it will be high since the developer who add it should understand the logic of the GraphicEditor.
- adding a new shape might affect the existing functionality in an undesired way, even if the new shape works perfectly

In order to have more dramatic effect, just imagine that the Graphic Editor is a big class, with a lot of functionality inside, written and changed by many developers, while the shape might be a class implemented only by one developer. In this case it would be great improvement to allow the adding of a new shape without changing the GraphicEditor class.

```
// Open-Close Principle - Bad example
class GraphicEditor {

    public void drawShape(Shape s) {
        if (s.m_type==1)
            drawRectangle(s);
        else if (s.m_type==2)
            drawCircle(s);
    }
    public void drawCircle(Circle r) {...}
    public void drawRectangle(Rectangle r) {...}
}

class Shape {
    int m_type;
}

class Rectangle extends Shape {
    Rectangle() {
        super.m_type=1;
    }
}

class Circle extends Shape {
    Circle() {
        super.m_type=2;
    }
}
```

```
}
```

Bellow is a example which supports the Open Close Principle. In the new design we use abstract draw() method in GraphicEditor for drawing objects, while moving the implementation in the concrete shape objects. Using the Open Close Principle the problems from the previous design are avoided, because GraphicEditor is not changed when a new shape class is added:

- no unit testing required.
- no need to understand the sourcecode from GraphicEditor.
- since the drawing code is moved to the concrete shape classes, it's a reduced risk to affect old functionality when new functionality is added.

```
// Open-Close Principle - Good example
class GraphicEditor {
    public void drawShape(Shape s) {
        s.draw();
    }
}

class Shape {
    abstract void draw();
}

class Rectangle extends Shape {
    public void draw() {
        // draw the rectangle
    }
}
Conclusion
```

Like every principle OCP is only a principle. Making a flexible design involves additional time and effort spent for it and it introduce new level of abstraction increasing the complexity of the code. So this principle should be applied in those area which are most likely to be changed.

There are many design patterns that help us to extend code without changing it. For instance the Decorator pattern help us to follow Open Close principle. Also the Factory Method or the Observer pattern might be used to design an application easy to change with minimum changes in the existing code.