Es un principio para el diseño orientado a objetos descrito por primera vez por Bertrand Meyer que dice que *"las entidades de software (clases, módulos, funciones, etc.) deben estar abiertas para la extensión, pero cerradas para la modificación"* .

Significa que **debemos esforzarnos por escribir código que no tenga que cambiarse cada vez que cambien los requisitos** . La forma en que hacemos eso puede diferir según el contexto. A menudo la solución implica herencia y polimorfismo, que es lo que ilustra el siguiente ejemplo cuya solución final es una jerarquía con clase abstracta que ya conocemos de un ejercicio anterior.

Ejemplo:

Trabajamos inicialmente sólo con la clase Rectangulo así que escribimos

```
class Rectangulo {
    double largo;
    double ancho;
    public Rectangulo(double largo, double ancho) {
        this.largo = largo;
        this.ancho = ancho;
    }

}
```

A continuación, nos solicitan una App capaz de calcular el área total de un conjunto de rectángulos y escribimos.

```
public class App{

    double area(Rectangulo[] rectangulos){
        double areaTotal=0;
        for(Rectangulo r:rectangulos){
            areaTotal+=r.ancho*r.largo;
        }
        return areaTotal;
    }
   public static void main(String[] args) {
     App miapp= new App();

     Rectangulo[] figuras={new Rectangulo(2,3), new Rectangulo(5,2)};
     System.out.println(miapp.area(figuras));
  }
}
```

A continuación nos dicen que está genial y si podríamos hacer lo mismo añadiendo también Circulos

```
class Circulo {
    double radio;

    public Circulo(double radio) {
        this.radio = radio;
    }
}
```

y entonces retocamos nuestra App de la siguiente manera.

```
public class App{

    double area(Object[] figuras){
        double areaTotal=0;
        for(Object o:figuras){
            if(o instanceof Rectangulo){
                Rectangulo r=(Rectangulo) o;
                areaTotal+=r.ancho*r.largo;
            }else{//es un circulo
                Circulo c=(Circulo) o;
                areaTotal+=c.radio*c.radio*Math.PI;
            }

        }
        return areaTotal;
    }
   public static void main(String[] args) {
     App miapp= new App();

     Object[] figuras={new Circulo(2), new Rectangulo(5,2)};
     System.out.println(miapp.area(figuras));
  }
}
```

Fue todo un éxito!! Y por eso … nos dicen que quieren también ¡Triangulos …!

Y ya nos damos cuenta de que algo no va bien, otro if más haciendo probablemente copiar y pegar de otro if y luego retocándolo y pienso "esto es una chapuza para un número indeterminado de tipos de figuras". Cada vez que me piden una nueva figura tengo que cambiar la App.

La solución correcta en este contexto cambiante es mover la responsabilidad del cálculo del área a cada figura y que cada tipo extienda a figura. Fíjate entonces que con esta nueva solución se pueden añadir nuevas figuras pero no hay que cambiar la App, es decir, **App está "cerrada" para modificación pero "abierta" para extensión a nuevos tipos.** Observa que limpio es el código del main() y sería igual aunque siguiéramos aumentando el tipo de figuras.

```
abstract class Figura{
    abstract double area();
}
class Rectangulo extends Figura{
    double largo;
    double ancho;
    public Rectangulo(double largo, double ancho) {
        this.largo = largo;
        this.ancho = ancho;
    }
    @Override
    double area() {
        return largo*ancho;
    }

}
class Circulo extends Figura{
    double radio;

    public Circulo(double radio) {
        this.radio = radio;
    }

    @Override
    double area() {
        return radio*radio*Math.PI;
    }
```

```
}

public class App{

    double area(Figura[] figuras){
        double areaTotal=0;
        for(Figura f:figuras){
            areaTotal+=f.area();
        }
        return areaTotal;
    }
   public static void main(String[] args) {
     App miapp= new App();

     Figura[] figuras={new Circulo(2), new Rectangulo(5,2)};
     System.out.println(miapp.area(figuras));
  }
}
```