# Módulo Entorno de desarrollo.
# 1º DAW.

PRÁCTICA2: Refactorización.
UNIDAD DE TRABAJO 5.
Profesor: Pedro Antonio Santiago Santiago.

# 1. Introducción.

Los desarrolladores, además de crear código realizan otra serie de funciones, muchas veces dedicando más tiempo y esfuerzo que a la de redactar código desde las pruebas vistas en el tema anterior, la optimización de diferentes partes del software como algoritmos o sentencias SQ, a refactorizar para hacer el código más mantenible, sin olvidar la documentación de los proyectos y código.

Está práctica se centra en el análisis y refactorización de código, para hacerlo más mantenible, comprensible y extensible.

# 2. Materiales.

1. Equipo con JVM, NetBeans y VSC instalados.

## 2.1. Cuestiones teóricas.

**Las cuestiones se resuelven a partir de los apuntes, no buscando en Internet. Si en alguna cuestión es necesario investigar se menciona en la misma.**

1. Define con tus palabras ¿Qué es y para qué se realiza la refactorización?
2. ¿En qué consiste la técnica de los dos sobreros? ¿Qué debe hacer cada uno de los sobreros? ¿Con un sobrero se pueden hacer cosas asignadas a otro sombrero?
3. ¿Qué es el “code smell? ¿Cómo encontrarlo?. Relacionar con la refactorización.
4. Si un fragmento de código no tiene utilidad ¿Qué tipo de “code Smell” es?
5. Se tiene un método de 250 líneas de código. Comentar si es candidato a ser “code smell” y las razones para ello.
6. Un método tiene 10 parámetros. Explicar si es no un “code smell”, y de ser así: la razón, como solucionarlo y que se obtiene al solucionarlo.
7. En la refactorización, si se encuentra una sentencia “switch”. ¿Puede ser un problema? ¿Cuáles son los posibles tratamientos en caso de serlo?
8. Se tiene un método muy largo, que es candidato a ser un “code smell”. ¿qué técnicas se utilizan para resolverlo? Explicarlas.

## 2.2. Ejercicios prácticos.

1. Se tiene el siguiente código:

```
public class Shape {
    public enum ShapeType{
        Circle,
        Rectangle,
        Square,
        Triangle
    }
    private ShapeType type;
    private int p1;
    private int p2;
    public Shape(ShapeType type,int p1,int p2) {
        this.type=type;
        this.p1=p1;
        this.p2=p2;
    }
    public double area(){
        double resultado=0.0d;
        switch (type) {
            case Circle:
                resultado =2*Math.PI*p1;
                break;
            case Rectangle:
            case Square:
                resultado= p1*p2;
                break;
            case Triangle:
                resultado= p1*p2/2;
            default:
                break;
        }
        return resultado;
    }
}
```

- Analizar si es candidato a la refactorización indicando los síntomas, la razón de que sea un problema, el tratamiento o solución, y los beneficios que se obtiene.
- Escribir el código para solucionarlo, usando los IDE's.

2. Se tiene un juego en el que el personaje puede tener diferentes elementos que le dan ciertas habilidades, por ejemplo, un escudo que decrementa el poder de ataque del enemigo en un 25%, o una espada que incrementa su poder de ataque en un 40%.

   El código es el siguiente:

```
public class Warrior {
    private double life;
    private String name;
    private double swordpower=1.0d;
    private String swordname="apprentice sword";
    private String shieldname="apprentice shield";
    private double shieldpower=0.25d;
    public Warrior(){
        this.life=100d;
        this.name="Undefined";
    }
    public Warrior(String name){
        this.name=name;
    }
    public double attack(){
        return this.swordpower;
    }
    public void defense(Warrior w){
        this.life-=w.attack()*shieldpower;
    }
    public String toString(){
        return "Name:"+this.name+" life:"+this.life+
"Shield:"+shieldname+" Sword:"+swordname;
    }

}
```

- ¿Qué sucede si ahora se tiene una espada que además de atacar también sirve para defenderse? ¿Y si se tiene un nuevo escudo con poderes mágicos que al defenderse en vez de decrementar la vida la incrementa?.
- Si se añaden nuevos escudos/espadas con cualidades nuevas a lo largo del proyecto, ¿es necesario estar cambiado y modificando el código?
- Buscar entre los “code smells” de tipo “change-preventers”, el que mejor casa con el problema anterior( justificar) y aplicar alguna de las técnicas propuestas (con código) usando las herramientas proporcionadas por los IDE’s (NetBeans o VSC). https://refactoring.guru/refactoring/smells/change-preventers

3. Un compañero ha realizado una prueba técnica para un trabajo y lo han descartado ya que su código huele mal. El no entiende la razón de que haya sido descartado para conseguir el empleo y nos pide que revisemos el código entregado para darle nuestra opinión. Aunque le han dicho que tiene que ver con el acoplamiento de las clases (https://refactoring.guru/refactoring/smells/couplers). Analizar el código y explicar la razón de que su código huela mal, decidir que técnica de refactorización usar para solucionarlo y qué se gana al realizar esta refactorización.

- Realizar la refactorización en el código usando los entornos de desarrollo.

```
public class Ship {
    private int x;
    private int y;
    public Bullet bullet;

    public Ship(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public void shoot() {
        this.bullet = new Bullet(x, y);

    }

    public void update() {
        this.bullet.setX(this.bullet.getX() + 1);
    }

    public void moveLeft() {
        this.x--;
    }

    public void moveRigth() {
        this.x++;
    }

    public void moveUp() {
        this.y--;
    }

    public void Down() {
        this.y++;
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

    public Bullet getBullet() {
```

```
        return bullet;
    }

    public void setBullet(Bullet bullet) {
        this.bullet = bullet;
    }

    public String toString(){
        return "X:"+this.x+" Y:"+this.y+"
Bullet>X:"+this.bullet.getX()+" Y:"+this.bullet.getY();
    }


    public boolean collision(Ship s) {
        if (s.getX() == this.bullet.getX() && s.getY() ==
this.bullet.getY())
            return true;
        else
            return false;
    }
}
```

```
public class Bullet {
    private int x;
    private int y;
    public Bullet(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public int getX() {
        return x;
    }
    public void setX(int x) {
        this.x = x;
    }
    public int getY() {
        return y;
    }
    public void setY(int y) {
        this.y = y;
    }
}
```

# 3. Entrega.

La práctica se entrega comprimida, con los proyectos limpios, y la memoria y preguntas en un fichero en formato PDF, en el campus virtual ww.aules.edu.gva.es. El documento de entrega ha de tener los siguientes puntos.

1. Portada. (Título de la práctica y autor).

2. Introducción.
3. Desarrollo de la práctica. **Capturas de pantalla, respuestas a las preguntas y comentarios.**
4. Conclusiones. (Pequeño comentario sobre la práctica: dificultad, problemas encontrados…).

# 4. Evaluación.

**Se puede realizar una corrección de forma presencial donde el profesor preguntará cuestiones sencillas sobre la práctica para comprobar la autoría de la misma.**

RA4 Optimiza código empleando las herramientas disponibles en el entorno de desarrollo.

CE4a. Se han identificado los patrones de refactorización más usuales.

CE4b. Se han elaborado las pruebas asociadas a la refactorización.

CE4e. Se han aplicado patrones de refactorización con las herramientas que proporciona el entorno de desarrollo.