18. Programación Orientada a Objetos(IV)

18.1 Clases abstractas.

Una clase abstracta es aquella que no va a tener instancias de forma directa, aunque sí habrá instancias de las subclases (siempre que esas subclases no sean también abstractas). Por ejemplo, si se define la clase Animal como abstracta, no se podrán crear objetos de la clase Animal, es decir, no se podrá hacer Animal mascota = new Animal(), pero sí se podrán crear instancias de la clase Gato, Ave o Pinguino que son subclases de Animal.

Se fija un conjunto de métodos y atributos que permitan modelar un cierto concepto, que será refinado mediante la herencia.

18.2 Métodos abstractos.

Solo cuentan con la declaración y no poseen cuerpo de definición.
La implementación es específica de cada subclase
• Toda clase que contenga algún método abstracto (heredado o no) es abstracta. Puede tener también métodos efectivos.
• Tiene que derivarse obligatoriamente
• No se puede hacer un new de una clase abstracta. Si deben definir los constructores.

```java
public abstract class Poligono {

    private String color;

    // Existe constructor aunque no se pueda crear una instancia
    // Las subclases pueden (y normalmente deben )llamar al constructor
    // de la superclase con super().

    public Poligono(String color) {
        this.color = color;
    }

    // Existen métodos concretos que se heredarán.

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    // No siempre podremos dar una implementación por defecto para que se herede
    // No existe una fórmula estándar de área o perímetro.

    public abstract double area();

    public abstract double perimetro();

    // El toString está llamando a area o perímetro pese a que no han sido implementadas
    // Cada subclase lo implementará, se ejecutará el comportamiento de la subclase
    // A este hecho se le llama polimorfismo.

    @Override
    public String toString() {
        return "Poligono [color=" + color + ", getColor()=" + getColor() + ", area()=" + area() + ", perimetro()="
                + perimetro() + ", getClass()=" + getClass() + ", hashCode()=" + hashCode() + ", toString()="
                + super.toString() + "]";
    }



}
```

```java
public class TrianguloEquilatero extends Poligono{


    private double base;
    private double altura;

    // Implementamos el método declarado en la clase abstracta
    // Si no lo implementamos no compila, salvo que lo declaremos abstracto
    // y lo implmenten sus subclases.


    public TrianguloEquilatero(String color, double base, double altura) {
        super(color);
        this.base = base;
        this.altura = altura;
    }


    public TrianguloEquilatero(String color, double lado) {
        super(color);
        this.base = lado;
        this.altura = Math.sqrt(3)*lado /2;
    }

    public double area() {
        return base * altura / 2;
    }


    public double getLado() {

        return Math.sqrt(altura*altura+base*base);
    }

    public void setLado(double lado) {
        this.base = lado;
        this.altura = Math.sqrt(3)*lado /2;
    }

    public double perimetro() {

        return 3 * this.getLado();
    }

    @Override
    public String toString() {
        return "TrianguloEquilatero [base=" + base + ", altura=" + altura + ", getColor()=" + getColor() + "]";
    }

}
```

```java
public class PentagonoRegular extends Poligono{


    private double lado;


    public PentagonoRegular(String color, double lado) {
        super(color);
        this.lado = lado;

    }

    public double getLado() {
        return lado;
    }

    public void setLado(double lado) {
        this.lado = lado;
    }


    @Override
    public double area() {
        return 1.72 * lado * lado;

    }

    @Override
    public double perimetro() {
        return 5 * lado;
    }

    @Override
    public String toString() {
        return "PentagonoRegular [lado=" + lado + ", getColor()=" + getColor() + "]";
    }



}
```

18.3 Introducción al polimorfismo.

Polimorfismo: En Programación Orientada a Objetos, se llama polimorfismo a la capacidad que tienen los objetos de distinto tipo (de distintas clases) de responder al mismo método.

Ligadura dinámica: en tiempo de ejecución se elegirá la versión mas adecuada según el tipo del objeto receptor.

```
public class CalculaPoligonos {

/*
 *  Calcula el area de un conjunto de polígonos
 */
    static public double area(Poligono[] poli) {
        double areaTotal = 0;
        for (Poligono p:poli) {
            //llama a area de Polígono, ejecutará el método de la subclase que sea !!!!
            areaTotal+=p.area();
        }

        return areaTotal;
    }

    /*
     *  Calcula el perímetro  de un conjunto de polígonos
     */
    static public double perimetro(Poligono[] poli) {
        double perimetroTotal = 0;
        for (Poligono p:poli) {
            //llama a area de Polígono, ejecutará el método de la subclase que sea !!!!
            perimetroTotal+=p.perimetro();
        }

        return perimetroTotal;
    }



    static public void main(String args[]) {

        TrianguloEquilatero t1 = new TrianguloEquilatero("Azul", 3);
        TrianguloEquilatero t2 = new TrianguloEquilatero("Verde", 4);
        PentagonoRegular p1 = new PentagonoRegular("Amarillo", 3);
        PentagonoRegular p2 = new PentagonoRegular("Amarillo", 5);

        // Definimos un array con distintas figuras.
        Poligono[] arrayPoli = {t1, t2, p1, p2};

        double sumaAreas = area(arrayPoli);
        double sumaPerimetros = perimetro(arrayPoli);

        System.out.println("Area total: " + sumaAreas);
        System.out.println("Perímetro total: " + sumaPerimetros);


    }
}
```

18.1. Crea los polígonos Rectángulo, Cuadrado y Hexágono.

18.2. Lanza una excepción en caso de que se intente crear un Polígono con un valor negativo.

18.3. Sube a una clase abstracta Personaje todo lo que hay en la clase Guerrero de nuestro miniRPG menos el método atacar que será abstracto e implementado en el Guerrero.

18.4 Implementar otra clase Dado. Por defecto el dado tendrá 6 caras. Tendremos tres constructores (uno al que no se le pasa nada e inicializa el dado al azar, otro al que sólo se le pasa que número tiene el dado en la cara superior y otro con el número del dado en la cara superior y el número de caras del dado). Implementa los getters, el método roll() que tirará el dado al azar y el toString(). Implementa un tester que tenga un vector de 4 dados y los lance una serie de veces.