7. CLASES ABSTRACTAS

Después de estudiar la herencia, vamos a analizar las clases abstractas Estas juegan un papel fundamental en otro de los conceptos clave de la POO: el **polimorfismo.**

7.1. Definición

Una clase abstracta es una clase en la que alguno de sus métodos está declarado pero no está definido, es decir, se especifica su nombre, parámetros *y* tipo de devolución pero no incluye código. A este tipo de métodos se les conoce como **métodos abstractos.**

Un método se define como abstracto porque en ese momento no se conoce como ha de ser su implementación, serán las subclases de la clase abstracta las responsables de darle "cuerpo" mediante la sobrescritura del mismo.

- **No es posible crear objetos de una clase abstracta.** Al haber métodos que no están definidos en la clase, no está permitido crear objetos de ella. Por ejemplo, si se declara la clase Figura como abstracta la siguiente instrucción no compilaría:

```
public abstract class Figura {

}
```

```
Figura f = new Figura(); // Error de compilación
```

Como ya se ha comentado, el objetivo de las clases abstractas es servir de base a futuras clases que, a través de la herencia, se encargarán de sobrescribir los métodos abstractos y darles sentido. La clase abstracta únicamente define el formato que tienen que tener ciertos métodos, dejando a las subclases los detalles de implementación de los mismos.

- **Las subclases de una clase abstracta están obligadas a sobrescribir todos los métodos abstractos que heredan.** En caso de que no interese sobrescribir alguno de esos métodos, la subclase deberá ser declarada también abstracta. El siguiente listado muestra un caso de una clase abstracta `Vehiculo`, de la que hereda la clase `Coche`, en la cual se sobrescribe el método abstracto `arrancar()`.

```
abstract class Vehiculo {
  public abstract void arrancar();
}

// La siguiente clase no compila
class Coche extends Vehiculo {
  public void arrancar(String s) {
    System.out.println("Arranca un coche" + " con " + s);
  }
}
```

Al intentar compilar la clase `Coche` obtendremos un error de compilación. El motivo es que la clase Coche, **no está sobrescribiendo** el método abstracto `arrancar()`, sino que **lo está sobrecargando al definir un nuevo método** `arrancar(String)`. Por tanto, la clase `Coche` seguirá teniendo un método `arrancar()` abstracto y a no ser que la clase `Coche` se declare como abstracta o se sobrescriba realmente el método `arrancar()` tal y como se indica en el siguiente listado, se producirá un error al intentar compilarla:

```
abstract class Vehiculo {
  public abstract void arrancar();
}

//Compila correctamente
class Coche extends Vehiculo {
  // sobrecarga del método arrancar
  public void arrancar(String s) {
    System.out.println("Arranca un coche" + " con " + s);
  }

  //sobrescitura del método arrancar() de la superclase
  @Override
  public void arrancar() {
    System.out.println ("Arranca un coche");
  }
}
```

- **Una clase abstracta puede tener constructores.** Aunque no es posible crear objetos de éstas, las clases abstractas serán heredadas por otras clases de las que sí se podrán crear objetos y, como sabemos, cuando se crea un objeto de una subclase se ejecuta también el constructor de la superclase. De hecho, al igual que sucede con una clase estándar, si no incluimos constructores explícitamente en la clase abstracta el compilador añadirá uno por defecto.

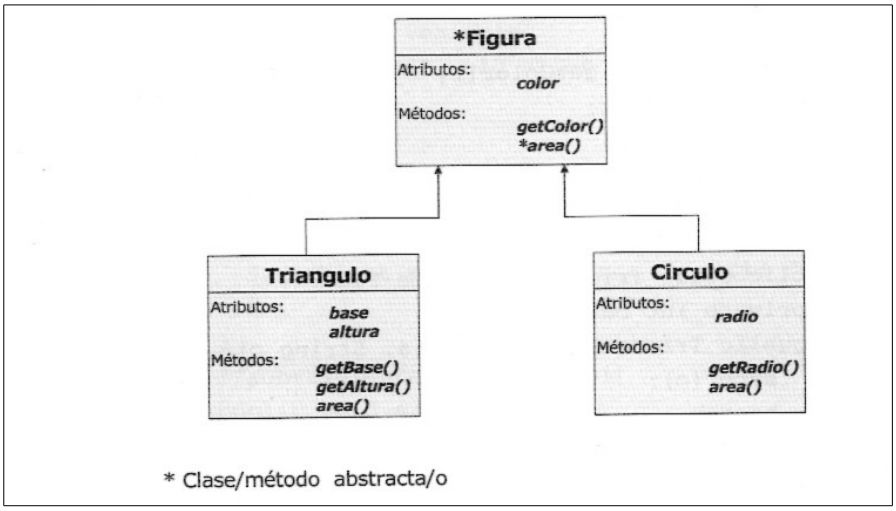
En el siguiente listado se muestra, por un lado, la clase `Figura` al completo, con sus atributos, constructores, métodos abstractos y métodos estándar. Por otro, están las subclases `Triangulo` y `Circulo` que proporcionan una implementación de los métodos abstractos de la clase `Figura`.

```
//Clase Figura
public abstract class Figura {

  private String color;

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

  public String getColor(){
    return this.color;
  }

  public abstract double area();
}
```

```
//Clase Circulo
public class Circulo extends Figura {

  private int radio;

  public Circulo(int radio, String color) {
    super(color);
    this.radio = radio ;
  }

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

  public int getRadio(){
    return this.radio;
  }
}
```

```java
//Clase Triangulo
public class Triangulo extends Figura {

  private int base, altura;

  public Triangulo(int base, int altura, String color) {
    super(color);
    this.base = base;
    this.altura = altura;
  }

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

  public int getBase(){
    return this.base;
  }

  public int getAltura(){
    return this.altura;
  }
}
```

La figura siguiente muestra una representación gráfica de estas clases y la relación entre ellas.



Clases Figura, Triangulo y Circulo.