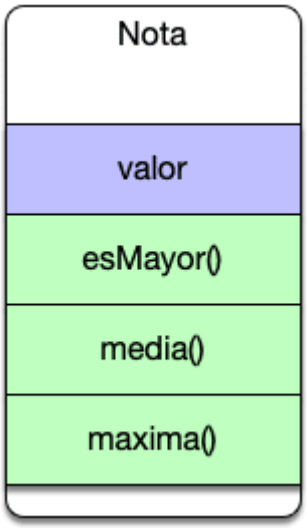
El uso de Java APIs es algo de lo más común cuando trabajamos en el día a día , pero en muchos casos no nos damos cuenta del gran esfuerzo que se ha realizado en su diseño para mantener un API sencillo y homogeneidad es algo que muchas veces se hecha en falta. Vamos a ver un ejemplo sencillo utilizando la clase Nota y varios métodos para hacer calculo sobre ella:

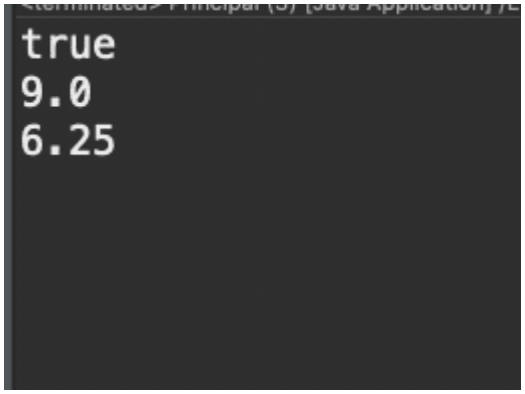



Como podemos observar en el diagrama la clase dispone de la propiedad valor y los métodos esMayor , media y máxima . Hasta aquí todo parece muy razonable y podemos utilizarlo en un programa main para trabajar con ello.

```
package com.arquitecturajava.ejemplo1;

public class Principal {

	public static void main(String[] args) {
		Nota nota1= new Nota (7);
		Nota nota2= new Nota (5);
		System.out.println(nota1.esMayor(nota2));
```

```
                Nota nota3= new Nota(4);
                Nota nota4= new Nota(9);
System.out.println(Nota.maxima(nota1,nota2,nota3,nota4));
System.out.println(Nota.media(nota1,nota2,nota3,nota4));
        }

}
```

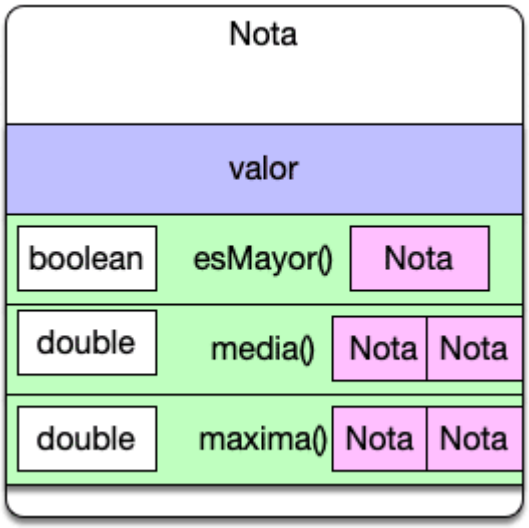
El resultado le podemos ver impreso en la consola y todo funciona perfectamente.



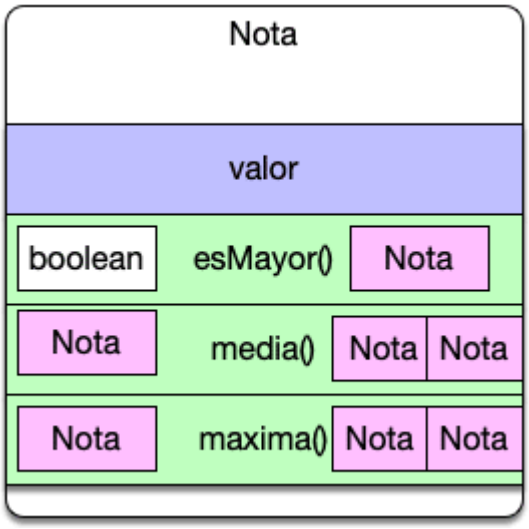

No hay mucho que objetar. ¿O tal vez sí?.

# Java APIs y Homegeneidad

La realidad es que la clase no es muy homogénea a la hora de trabajar con el concepto de Nota ya que recibe en muchos casos la variable Nota y siempre devuelve un valor numérico a la hora de utilizarla.




Si trabajamos más el concepto de homegeneidad podremos conseguir una clase más sólida y con mejor reutilización. Para ello modificaremos los métodos media y máxima que serán los encargados de devolver una nueva nota con la media o una nota con el máximo valor mejorando la flexibilidad




El código quedaría:

```
package com.arquitecturajava.ejemplo2;
```

```
public class Nota {

	private double valor;

	public double getValor() {
		return valor;
	}

	public void setValor(double valor) {
		this.valor = valor;
	}

	public Nota(double valor) {
		super();
		this.valor = valor;
	}

	public boolean esMayor(Nota nota) {

		if (this.getValor() > nota.getValor()) {
			return true;

		} else {
			return false;
		}
	}

	public  static Nota media(Nota... notas) {
		double total = 0;

		for (Nota nota : notas) {
```

```
                        total += nota.getValor();
                }
                return new Nota(total / notas.length);
        }

        public static Nota maxima(Nota... notas) {

                Nota maxima = notas[0];

                for (Nota nota : notas) {

                        if (nota.getValor() > maxima.getValor()) {

                                maxima = nota;
                        }

                }
                return maxima;
        }

}
```

Esto nos facilitaría por ejemplo el operar con Notas y calcular la Nota máxima de las medias:

```
package com.arquitecturajava.ejemplo2;

public class Principal {

        public static void main(String[] args) {
                Nota nota1= new Nota (7);
                Nota nota2= new Nota (5);
```

```
                System.out.println(nota1.esMayor(nota2));

                Nota nota3= new Nota(4);
                Nota nota4= new Nota(9);
                Nota media1=Nota.media(nota1,nota2);
                Nota media2= Nota.media(nota3,nota4);
System.out.println(Nota.maxima(media1,media2).getValor());
        }

}
```

El resultado es que la media más alta es es la media 2 con 6.5:

## Conclusión

Apliquemos el concepto de homogeneidad al diseñar nuestras Java APIs y conseguiremos ganar en flexibilidad y reutilización:


## Otros artículos relacionados

- Java Interfaces y el concepto de simplicidad
- Java Herencia vs Interfaces
- java == null vs Objects.isNull y sus diferencias
- Java Polimorfismo, Herencia y simplicidad
- https://en.wikipedia.org/wiki/List_of_Java_APIs