

Constructor de copia

Es un constructor que se usa para crear un objeto a partir de otro del mismo tipo.

lo defino :

- Cuando quiero hacer copias profundas de objetos inmutables.
- Cuando no quiero que se copien todos los campos.

Si tenemos un puntero lo copia como tal No copia a lo que apunta sino que guarda la misma dirección de memoria.

Esto puede traer problemas ya que podemos:

- Modificar el mismo dato desde dos objetos
- Perder el acceso al dato al que se apuntaba.

Copia superficial

```
class Alumno { 1 usage
    private String nombre; 3 usages
    private int[] notas; 3 usages

    public Alumno(String nombre, int[] notas) { no usages
        this.nombre = nombre;
        this.notas = notas; // referencia compartida
    }

    public Alumno(Alumno otro) { no usages
        this.nombre = otro.nombre;
        this.notas = otro.notas;
    }
}
```

Copia Profunda

```
class Alumno { 1 usage
    private String nombre; 3 usages
    private int[] notas; 6 usages

    public Alumno(String nombre, int[] notas) { no usages
        this.nombre = nombre;
        this.notas = notas;
    }

    public Alumno(Alumno otro) { no usages
        this.nombre = otro.nombre;
        this.notas = new int[otro.notas.length];
        for (int i = 0; i < otro.notas.length; i++) {
            this.notas[i] = otro.notas[i];
        }
    }
}
```


Sobre carga es el poder usar el "mismo método" para que se comporte según corresponda dependiendo de los parámetros que recibe.

```
class Calculadora { no usages
    public int sumar(int a, int b) { no usages
        return a + b;
    }

    public double sumar(double a, double b) { no usages
        return a + b;
    }

    public int sumar(int a, int b, int c) { no usages
        return a + b + c;
    }
}
```

Por fuera el uso es siempre llamando a sumar por lo que para la persona no hay un trato especial, pero cada método tiene su propia implementación permitiendo la flexibilidad de adaptar el caso al tipo o cantidad de parámetros como se requiera.

TDA Vector

Es una estructura guardada en bloques de memoria de forma continua que almacena elementos del mismo tipo y tiene como funciones:

- Guardar
- Obtener
- Buscar
- Eliminar

Vector con memoria dinámica

Si quisiéramos crear un vector de enteros en forma dinámica podemos solicitar todo un bloque al heap con new:

```
int[] vector = new int[100];
```

En la instrucción anterior estamos creando un bloque contiguo de 100 enteros del cual guardamos la referencia a ese bloque en la variable.

Ya vimos cómo podemos acceder a cada uno de esos enteros, trabajando con la variable puntero como si fuera un vector común y corriente.

Redimensión

Cuando no sabemos cuántos elementos vamos a guardar no es conveniente hacer un vector de "espacio infinito" sino usar memoria dinámica y redimensionar el mismo, es decir, pedir más memoria a medida que sea necesario.

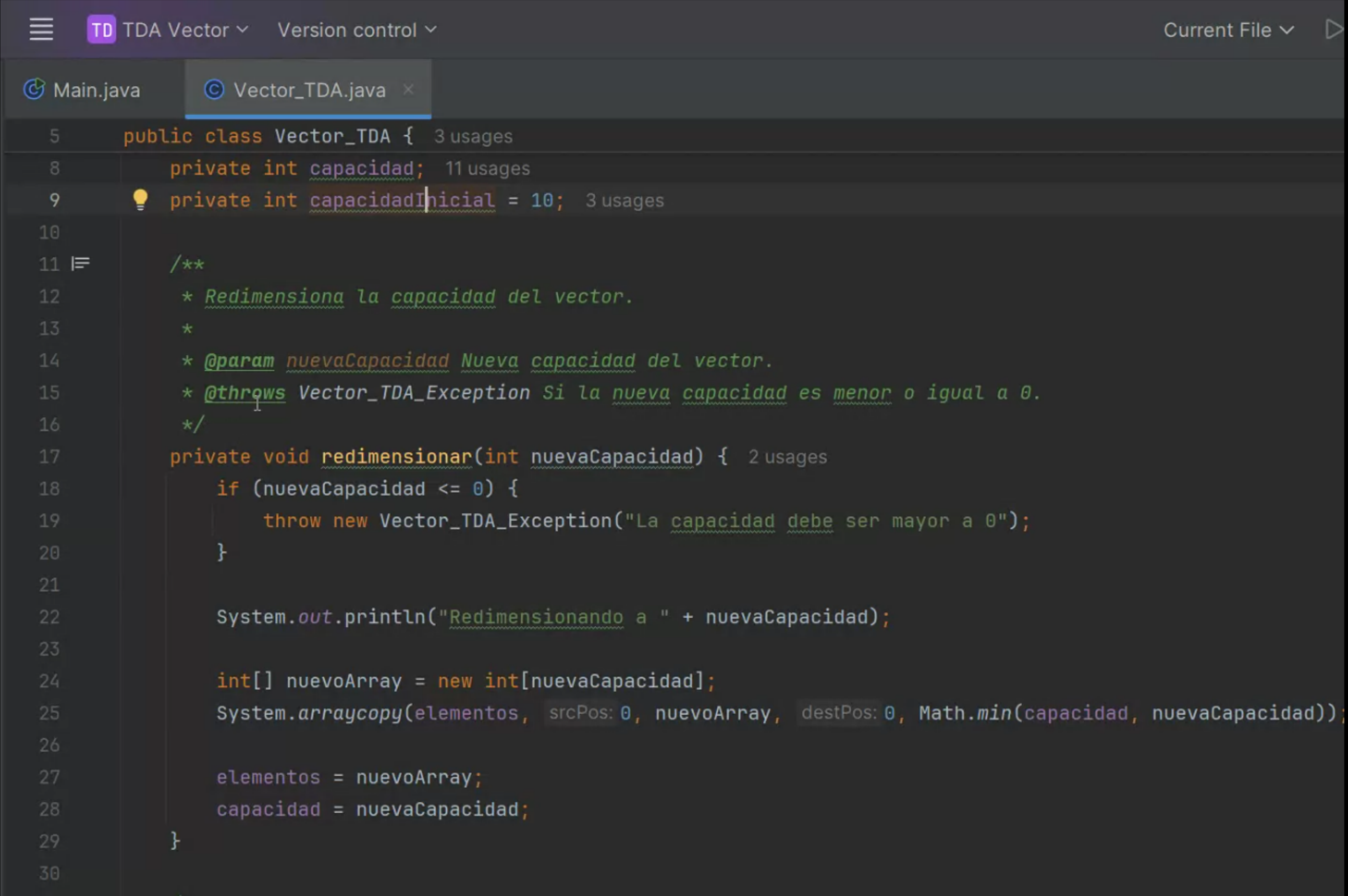


```
class Alumno { 1 usage
    private String nombre; 3 usages
    private int[] notas; 6 usages

    public Alumno(String nombre, int[] notas) { no usages
        this.nombre = nombre;
        this.notas = notas;
    }

    public Alumno(Alumno otro) { no usages
        this.nombre = otro.nombre;
        this.notas = new int[otro.notas.length];
        for (int i = 0; i < otro.notas.length; i++) {
            this.notas[i] = otro.notas[i];
        }
    }
}
```


Escribir en código o pseudocódigo un método de redimensión de un vector.



```
5 public class Vector_TDA { 3 usages
8     private int capacidad; 11 usages
9     private int capacidadInicial = 10; 3 usages
10
11 /**
12  * Redimensiona la capacidad del vector.
13  *
14  * @param nuevaCapacidad Nueva capacidad del vector.
15  * @throws Vector_TDA_Exception Si la nueva capacidad es menor o igual a 0.
16  */
17 private void redimensionar(int nuevaCapacidad) { 2 usages
18     if (nuevaCapacidad <= 0) {
19         throw new Vector_TDA_Exception("La capacidad debe ser mayor a 0");
20     }
21
22     System.out.println("Redimensionando a " + nuevaCapacidad);
23
24     int[] nuevoArray = new int[nuevaCapacidad];
25     System.arraycopy(elementos, srcPos: 0, nuevoArray, destPos: 0, Math.min(capacidad, nuevaCapacidad));
26
27     elementos = nuevoArray;
28     capacidad = nuevaCapacidad;
29 }
30
31 /**
```