Universidad Católica de Santa María

FACULTAD DE CIENCIAS E INGENIERÍAS FÍSICAS Y FORMALES

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

ASIGNATURA:

LENGUAJES DE PROGRAMACIÓN III - PRÁCTICA GRUPO 04

DOCENTE:

MARIO SANTILLANA VALDIVIA

TEMA:

FICHA 10 – METODOS GENERICOS

PRESENTADO POR:

GARZÓN MENDOZA ANA PAULA VIVIANA

TICONA CUNO ADRIANA MILAGROS

AREQUIPA-PERÚ

2023

CÓDIGO	APELLIDOS Y NOMBRES	FECHA
2022221882	GARZÓN MENDOZA ANA PAULA VIVIANA	25/10/2023
2022802702	TICONA CUNO ADRIANA MILAGROS	25/10/2023

Actividades

1. Implementar el código de los ejemplos visto en el marco teórico para poder apreciar los resultados devueltos y analizar el funcionamiento el código. Para el caso del ejercicio del método imprimirArray añade el código necesario para verificar si funciona con arreglos de tipo Float, de tipo Boolean y de tipo Persona (esta última clase es una creada por usted con mínimo los atributos DNI, nombres y dirección).

Persona.java > Persona > main(String[])

```java
//ACTIVIDAD 1
public class Persona {
    private String dni;
    private String nombres;
    private String direccion;

    public Persona(String dni, String nombres, String direccion) {
        this.dni = dni;
        this.nombres = nombres;
        this.direccion = direccion;
    }

    @Override
    public String toString() {
        return "Persona[" + "dni= " + dni + " , nombres= " + nombres  + " , direccion= " + direccion + ']';
    }
    public static <T> void imprimirArray(T[] array) {
        for (T elemento : array) {
            System.out.println(elemento);
        }
    }
    Run | Debug
    public static void main(String[] args) {
        // Float
        Float[] arrayFloat = {1.5f, 2.5f, 3.5f};
        System.out.println(x:"Array de Float:");
        imprimirArray(arrayFloat);

        // Boolean
        Boolean[] arrayBoolean = {true, false, true};
        System.out.println(x:"\nArray de Boolean:");
        imprimirArray(arrayBoolean);

        // Persona
        Persona[] arrayPersona = {
                new Persona(dni:"12345678", nombres:"Juan Perez", direccion:"Calle A"),
                new Persona(dni:"87654321", nombres:"Maria Rodriguez", direccion:"Calle B"),
                new Persona(dni:"01890998", nombres:"Pedro Gomez", direccion:"Calle C")
        };
        System.out.println(x:"\nArray de Personas:");
        imprimirArray(arrayPersona);
    }
}
```

```
PS C:\Users\Ana Paula\OneDrive\UNIVERSIDAD\LP 3\PRACTICA\FICHA 10>  & 'C:\Pr
e\User\workspaceStorage\d50e20a4fc1481d8ae22b4481fc5b299\redhat.java\jdt_ws\
Array de Float:
1.5
2.5
3.5

Array de Boolean:
true
false
true

Array de Personas:
Persona[dni= 12345678 , nombres= Juan Perez , direccion= Calle A]
Persona[dni= 87654321 , nombres= Maria Rodriguez , direccion= Calle B]
Persona[dni= 01890998 , nombres= Pedro Gomez , direccion= Calle C]
PS C:\Users\Ana Paula\OneDrive\UNIVERSIDAD\LP 3\PRACTICA\FICHA 10>
```

2. Escriba un método mínimo ( ) que reciba 4 parámetros de tipo genérico y devuelva el menor de los 4. Verifique su Funcionamiento con tipos Integer, Double, String y Persona (en este último caso, el criterio de orden entre personas se establece por los nombres – orden alfabético).

```java
//ACTIVIDAD 2
class Persona2 implements Comparable<Persona2> {
    private String dni;
    private String nombres;
    private String direccion;

    public Persona2(String dni, String nombres, String direccion) {
        this.dni = dni;
        this.nombres = nombres;
        this.direccion = direccion;
    }

    @Override
    public int compareTo(Persona2 otraPersona) {
        // Comparación basada en los nombres
        return this.nombres.compareTo(otraPersona.nombres);
    }

    @Override
    public String toString() {
        return "Persona2[" + "dni=" + dni + " , nombres= " + nombres + " , direccion= " + direccion + ']';
    }
    public static <T extends Comparable<T>> T minimo(T a, T b, T c, T d) {
        T min = a;

        if (b.compareTo(min) < 0) {
            min = b;
        }

        if (c.compareTo(min) < 0) {
            min = c;
        }

        if (d.compareTo(min) < 0) {
            min = d;
        }

        return min;
    }

    Run | Debug
    public static void main(String[] args) {
        // Integer
        Integer minInt = minimo(a:5, b:2, c:8, d:1);
        System.out.println("Mínimo Integer: " + minInt);

        // Double
        Double minDouble = minimo(a:3.5, b:1.2, c:4.7, d:2.1);
        System.out.println("Mínimo Double: " + minDouble);

        // String
        String minString = minimo(a:"perro", b:"gato", c:"elefante", d:"ardilla");
        System.out.println("Mínimo String: " + minString);

        // Persona2
        Persona2 minPersona = minimo(
                new Persona2(dni:"12345678", nombres:"Juan Perez", direccion:"Urb. A"),
                new Persona2(dni:"87654321", nombres:"Maria Rodriguez", direccion:"Urb. B"),
                new Persona2(dni:"55555555", nombres:"Pedro Gomez", direccion:"Urb. C"),
                new Persona2(dni:"98765432", nombres:"Ana Garcia", direccion:"Urb. D")
        );
        System.out.println("Mínimo Persona2: " + minPersona);
    }
}
```

```
PS C:\Users\Ana Paula\OneDrive\UNIVERSIDAD\LP 3\PRACTICA\FICHA 10>  & 'C:\Program File
Ana Paula\AppData\Roaming\Code\User\workspaceStorage\d50e20a4fc1481d8ae22b4481fc5b299\
Mínimo Integer: 1
Mínimo Double: 1.2
Mínimo String: ardilla
Mínimo Persona2: Persona2[dni=98765432 , nombres= Ana Garcia , direccion= Urb. D]
PS C:\Users\Ana Paula\OneDrive\UNIVERSIDAD\LP 3\PRACTICA\FICHA 10>
```

3. Sobrecargar el método imprimirArray visto anteriormente en el marco teórico para que este pueda tomar 2 argumentos adicionales de tipo entero, límite Inferior y limite Superior. Una llamada a este método sólo imprimirá una porción designada del arreglo. Validar límite Inferior y limite Superior. Si cualquiera de ellos está fuera de rango, el método sobrecargado imprimirArray debe lanzar una excepción LimiteInvalidoException (ésta es una excepción creada por usted), de lo contrario deberá retornar la impresión del número de elementos que se encuentran en el rango determinado.

```java
// ACTIVIDAD 3
public class LimInvalidoEx extends Exception {
    public LimInvalidoEx(String mensaje) {
        super(mensaje);
    }

    Run | Debug
    public static void main(String[] args) {
        // int
        Integer[] arrayInt = {1, 2, 3, 4, 5};
        try {
            System.out.println(x:"Elementos en el rango: ");
            int numElementos = imprimirArray(arrayInt, limiteInferior:0, limiteSuperior:3);
            System.out.println("Numero de elementos impresos: " + numElementos);
        } catch (LimInvalidoEx e) {
            System.err.println("Error: " + e.getMessage());
        }

        // string
        String[] arrayString = {"a", "b", "c", "d", "e"};
        try {
            System.out.println(x:"Elementos en el rango: ");
            int numElementos = imprimirArray(arrayString, limiteInferior:0, limiteSuperior:2);
            System.out.println("Numero de elementos impresos: " + numElementos);
        } catch (LimInvalidoEx e) {
            System.err.println("Error: " + e.getMessage());
        }
    }

    public static <T> int imprimirArray(T[] array, int limiteInferior, int limiteSuperior) throws LimInvalidoEx {
        if (limiteInferior < 0 || limiteSuperior >= array.length || limiteInferior > limiteSuperior) {
            throw new LimInvalidoEx(mensaje:"Limites de invalidos");
        }

        for (int i = limiteInferior; i <= limiteSuperior; i++) {
            System.out.println(array[i]);
        }

        return limiteSuperior - limiteInferior + 1;
    }
}
```

```
PS C:\Users\Ana Paula\OneDrive\UNIVERSIDAD\LP 3\PRACTICA\FICHA 10>  c:;
nMessages' '-cp' 'C:\Users\Ana Paula\AppData\Roaming\Code\User\workspace
Elementos en el rango:
1
2
3
4
Numero de elementos impresos: 4
Elementos en el rango:
a
b
c
Numero de elementos impresos: 3
PS C:\Users\Ana Paula\OneDrive\UNIVERSIDAD\LP 3\PRACTICA\FICHA 10>  c:;
nMessages' '-cp' 'C:\Users\Ana Paula\AppData\Roaming\Code\User\workspace
Elementos en el rango:
1
2
3
4
Numero de elementos impresos: 4
Elementos en el rango:
Error: Limites de invalidos
PS C:\Users\Ana Paula\OneDrive\UNIVERSIDAD\LP 3\PRACTICA\FICHA 10>
```

Ejercicios

1. Sobrecargar el método Genérico imprimirArray con un método No Genérico que específcamente imprimirá un array de tipo String en formato tabular ordenado, tal como se muestra en la imagen a continuación. (Implemente una aplicación para probar que funciona)

Array de cadenas contiene:

Uno Dos Tres Cuatro

Cinco Seis Siete Ocho

```
public class ImpresorArray {

    // Método genérico para imprimir cualquier tipo de array
    public static <T> void imprimirArray(T[] array) {
        for (T elemento : array) {
            System.out.print(elemento + " ");
        }
        System.out.println();
    }

    // Método no genérico específico para imprimir arrays de tipo String en formato tabular
    public static void imprimirArray(String[] array) {
        for (String elemento : array) {
            System.out.printf(format:"%-10s", args: elemento);
        }
        System.out.println();
    }

    public static void main(String[] args) {
        // Ejemplo de uso con un array de cadenas
        String[] arrayCadenas = {"Uno", "Dos", "Tres", "Cuatro", "Cinco", "Seis", "Siete", "Ocho"};

        System.out.println(x: "Array de cadenas contiene:");
        // Llamada al método genérico para imprimir cualquier tipo de array
        imprimirArray(array: arrayCadenas);

        // Llamada al método específico para imprimir arrays de tipo String en formato tabular
        imprimirArray(array: arrayCadenas);
    }
}
```

```
Output - SESION10 (run)
    run:
    Array de cadenas contiene:
    Uno       Dos       Tres      Cuatro    Cinco     Seis      Siete     Ocho
    Uno       Dos       Tres      Cuatro    Cinco     Seis      Siete     Ocho
    BUILD SUCCESSFUL (total time: 1 second)
```

2. Escribir un método genérico que intercambie las posiciones de 2 elementos diferentes en un arreglo. (Implemente una aplicación para probar que funciona)

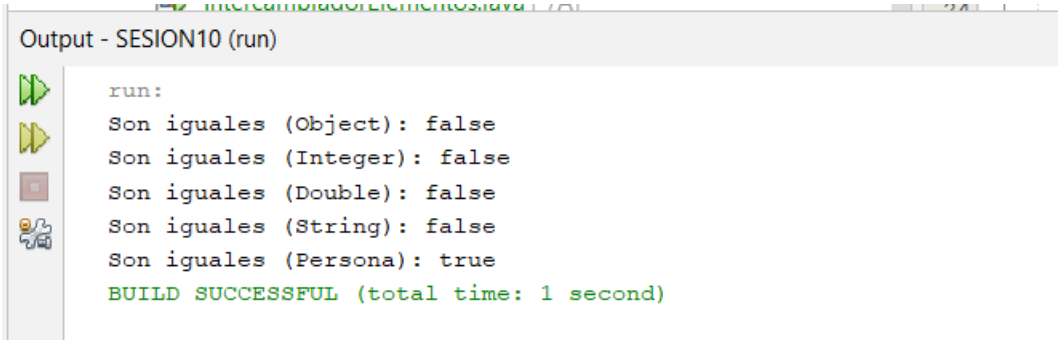
```java
public class IntercambiadorElementos {

    // Método genérico para intercambiar las posiciones de dos elementos en un arreglo
    public static <T> void intercambiarElementos(T[] array, int indice1, int indice2) {
        if (indice1 >= 0 && indice1 < array.length && indice2 >= 0 && indice2 < array.length) {
            T temp = array[indice1];
            array[indice1] = array[indice2];
            array[indice2] = temp;
        } else {
            System.out.println(x: "Índices fuera de rango.");
        }
    }

    public static void main(String[] args) {
        // Ejemplo de uso con un array de enteros
        Integer[] arrayEnteros = {1, 2, 3, 4, 5};

        System.out.println("Array original: " + Arrays.toString(a: arrayEnteros));

        // Intercambiar los elementos en las posiciones 1 y 3
        intercambiarElementos(array: arrayEnteros, indice1: 4, indice2: 2);

        System.out.println("Array después del intercambio: " + Arrays.toString(a: arrayEnteros));
    }
}
```

```
Output - SESION10 (run)
run:
Array original: [1, 2, 3, 4, 5]
Array despu?s del intercambio: [1, 2, 5, 4, 3]
BUILD SUCCESSFUL (total time: 1 second)
```

3. Escribir una versión genérica simple del método isEqualTo que compare 2 argumentos con el metodo equals y retorne true si son iguales y false en caso contrario. Usa este método genérico en un programa que llame al método isEqualsTo con una variedad de tipos como Object, Integer, Double, String, etc. ¿Qué resultado se obtiene cuando intentas ejecutar este programa? (Implemente una aplicación para probar que funciona)

```java
public class ComparadorGenerico {
    // Método genérico para comparar dos objetos utilizando el método equals
    public static <T> boolean isEqualTo(T obj1, T obj2) {
        // Si ambos objetos son nulos, son iguales
        if (obj1 == null && obj2 == null) {
            return true;
        }
        // Si uno de los objetos es nulo,  no son iguales
        if (obj1 == null || obj2 == null) {
            return false;
        }
        // metodo equals para comparar
        return obj1.equals(obj:obj2);
    }
    public static void main(String[] args) {
        // uso con diferentes tipos de datos
        Object objeto1 = "Hola";
        Object objeto2 = "Hol";
        System.out.println("Son iguales (Object): " + isEqualTo(obj1: objeto1, obj2: objeto2));

        Integer numero1 = 5;
        Integer numero2 = 6;
        System.out.println("Son iguales (Integer): " + isEqualTo(obj1: numero1, obj2: numero2));

        Double decimal1 = 3.14;
        Double decimal2 = 3.15;
        System.out.println("Son iguales (Double): " + isEqualTo(obj1: decimal1, obj2: decimal2));

        String cadena1 = "Arroz";
        String cadena2 = "Zanahoria";
        System.out.println("Son iguales (String): " + isEqualTo(obj1: cadena1, obj2: cadena2));

        Persona persona1 = new Persona(nombre:"Jose", edad: 30);
        Persona persona2 = new Persona(nombre:"Jose", edad: 30);
        System.out.println("Son iguales (Persona): " + isEqualTo(obj1: persona1, obj2: persona2));
    }
}
```

```java
public class Persona {
    private String nombre;
    private int edad;
    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public int getEdad() {
        return edad;
    }
    public void setEdad(int edad) {
        this.edad = edad;
    }
    // Método equals para comparar personas
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null || getClass() != obj.getClass()) {
            return false;
        }
        Persona persona = (Persona) obj;
        return edad == persona.edad && nombre.equals(anObject: persona.nombre);
    }
}
```



4. Escriba un método genérico que permita buscar un elemento cualquier en un arreglo de elementos. En caso existe el elemento buscado se debe devolver la posicion que ocupa, sino debe de devolver -1.

Para probar el método del ejercicio anterior, cree las siguientes clases:

a. Producto : codigo, descripcion, precio

b. Persona: dni, nombres, direccion

c. Estudiante: (derivada de Persona), tiene además, la carrera que estudia.

Las búsquedas que deberá realizar son por los siguientes atributos:

a. Producto: por su codigo

b. Persona: por su nombres

c. Estudiante: por sus nombres y carrera.

```java
public class BuscadorElemento {
    // Método genérico para buscar un elemento en un arreglo
    public static <T> int buscarElemento(T[] arreglo, String atributoBuscado, String valorBuscado) {
        for (int i = 0; i < arreglo.length; i++) {
            T elemento = arreglo[i];
            switch (atributoBuscado) {
                case "codigo":
                    if (elemento instanceof Producto) {
                        Producto producto = (Producto) elemento;
                        if (producto.getCodigo().equals(anObject: valorBuscado)) {
                            return i;
                        }
                    }
                    break;
                case "nombres":
                    if (elemento instanceof Persona) {
                        Persona persona = (Persona) elemento;
                        if (persona.getNombres().equals(anObject: valorBuscado)) {
                            return i;
                        }
                    }
                    break;
                case "nombresCarrera":
                    if (elemento instanceof Estudiante) {
                        Estudiante estudiante = (Estudiante) elemento;
                        if (estudiante.getNombres().equals(anObject: valorBuscado) && estudiante.getCarrera().equals(anObjec
                            return i;
                        }
                    }
                    break;
                default:
                    throw new IllegalArgumentException(s: "Atributo de búsqueda no válido");
            }
        }
        return -1; // Si no se encuentra el elemento, devolver -1
    }
```

```java
class Estudiante extends Persona {
    private final String carrera;

    public Estudiante(String dni, String nombres, String direccion, String carrera) {
        super(dni, nombres, direccion);
        this.carrera = carrera;
    }

    public String getCarrera() {
        return carrera;
    }
}

class Persona {
    private final String dni;
    private final String nombres;
    private final String direccion;

    public Persona(String dni, String nombres, String direccion) {
        this.dni = dni;
        this.nombres = nombres;
        this.direccion = direccion;
    }

    public String getNombres() {
        return nombres;
    }
}

class Producto {
    private final String codigo;
    private final String descripcion;
    private final double precio;

    public Producto(String codigo, String descripcion, double precio) {
        this.codigo = codigo;
        this.descripcion = descripcion;
        this.precio = precio;
    }

    public String getCodigo() {
        return codigo;
    }
}
```

```java
public class NewClass3 {
    public static void main(String[] args) {
        // Ejemplo de uso con diferentes tipos de datos
        Producto[] productos = {
                new Producto(codigo:"001", descripcion: "Producto1", precio:10.0),
                new Producto(codigo:"002", descripcion: "Producto2", precio:20.0),
                new Producto(codigo:"003", descripcion: "Producto3", precio:30.0)
        };

        Persona[] personas = {
                new Persona(dni:"123", nombres: "Persona1", direccion:"Dirección1"),
                new Persona(dni:"456", nombres: "Persona2", direccion:"Dirección2"),
                new Persona(dni:"789", nombres: "Persona3", direccion:"Dirección3")
        };

        Estudiante[] estudiantes = {
                new Estudiante(dni:"999", nombres: "Estudiante1", direccion:"Dirección4", carrera: "Carrera1"),
                new Estudiante(dni:"888", nombres: "Estudiante2", direccion:"Dirección5", carrera: "Carrera2"),
                new Estudiante(dni:"777", nombres: "Estudiante3", direccion:"Dirección6", carrera: "Carrera3")
        };

        // Búsqueda por código en productos
        int posicionProducto = buscarElemento(arreglo:productos, atributoBuscado: "codigo", valorBuscado: "001");
        System.out.println("Posición del producto: " + posicionProducto);

        // Búsqueda por nombres en personas
        int posicionPersona = buscarElemento(arreglo:personas, atributoBuscado: "nombres", valorBuscado: "Persona3");
        System.out.println("Posición de la persona: " + posicionPersona);

        // Búsqueda por nombres y carrera en estudiantes
        int posicionEstudiante = buscarElemento(arreglo:estudiantes, atributoBuscado: "nombresCarrera", valorBuscado: "Estudiante2")
        System.out.println("Posición del estudiante: " + posicionEstudiante);
    }
}
```

Output - SESION10 (run)

```
run:
Posici▯n del producto: 0
Posici▯n de la persona: 2
Posici▯n del estudiante: -1
BUILD SUCCESSFUL (total time: 1 second)
```

5. Modifique el método del ejercicio 4, para que sólo pueda buscar en arreglos que contengan objetos numéricos (aplique la genericidad restringida). Evidencia lo realizado con varias pruebas.

```java
public class BuscadorElemento {

    // Método genérico para buscar un elemento en un arreglo
    public static <T extends Number> int buscarElemento(T[] arreglo, String atributoBuscado, String valorBuscado) {
        for (int i = 0; i < arreglo.length; i++) {
            T elemento = arreglo[i];
            switch (atributoBuscado) {
                case "codigo":
                    // Validación para asegurar que el elemento sea una instancia de Number
                    if (elemento instanceof Producto) {
                        Producto producto = (Producto) elemento;
                        if (producto.getCodigo().equals(anObject: valorBuscado)) {
                            return i;
                        }
                    }
                    break;
                // Otros casos de búsqueda según el atributo
                default:
                    throw new IllegalArgumentException(s: "Atributo de búsqueda no válido");
            }
        }
        return -1; // Si no se encuentra el elemento, devolver -1
    }

}
```

```java
public class NewClass3 {
    public static void main(String[] args) {
        // Ejemplo de uso con diferentes tipos de datos
        Producto[] productos = {
                new Producto(codigo:"001", descripcion: "Producto1", precio:10.0),
                new Producto(codigo:"002", descripcion: "Producto2", precio:20.0),
                new Producto(codigo:"003", descripcion: "Producto3", precio:30.0)
        };

        Persona[] personas = {
                new Persona(dni:"123", nombres: "Persona1", direccion:"Dirección1"),
                new Persona(dni:"456", nombres: "Persona2", direccion:"Dirección2"),
                new Persona(dni:"789", nombres: "Persona3", direccion:"Dirección3")
        };

        Estudiante[] estudiantes = {
                new Estudiante(dni:"999", nombres: "Estudiante1", direccion:"Dirección4", carrera: "Carrera1"),
                new Estudiante(dni:"888", nombres: "Estudiante2", direccion:"Dirección5", carrera: "Carrera2"),
                new Estudiante(dni:"777", nombres: "Estudiante3", direccion:"Dirección6", carrera: "Carrera3")
        };

        // Búsqueda por código en productos
        int posicionProducto = buscarElemento(arreglo: productos, atributoBuscado: "codigo", valorBuscado: "001");
        System.out.println("Posición del producto: " + posicionProducto);

        // Búsqueda por nombres en personas
        int posicionPersona = buscarElemento(arreglo: personas, atributoBuscado: "nombres", valorBuscado: "Persona3");
        System.out.println("Posición de la persona: " + posicionPersona);

        // Búsqueda por nombres y carrera en estudiantes
        int posicionEstudiante = buscarElemento(arreglo: estudiantes, atributoBuscado: "nombresCarrera", valorBuscado: "Estudiante2")
        System.out.println("Posición del estudiante: " + posicionEstudiante);
```

```
Output - SESION10 (run)
run:
Posici?n del producto: -1
Posici?n de la persona: -1
Posici?n del estudiante: -1
BUILD SUCCESSFUL (total time: 1 second)
```

6. Escribir un método genérico copyArray() que reciba dos arreglos de tipo genérico y retorne un único array que contenga los elementos de ambos arreglos sin repetidos. Pruebe su funcionamiento con arreglos de tipo String, Producto y Persona.

```java
import java.util.Objects;
class Producto {
    private String nombre;
    private double precio;

    public Producto(String nombre, double precio) {
        this.nombre = nombre;
        this.precio = precio;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Producto producto = (Producto) obj;
        return nombre.equals(producto.nombre);
    }

    @Override
    public int hashCode() {
        return Objects.hash(nombre);
    }

    @Override
    public String toString() {
        return "Producto{" +
                "nombre='" + nombre + '\'' +
                ", precio=" + precio +
                '}';
    }
}
```

```java
import java.util.Objects;
class Persona {
    private String nombre;
    private int edad;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Persona persona = (Persona) obj;
        return edad == persona.edad && nombre.equals(persona.nombre);
    }

    @Override
    public int hashCode() {
        return Objects.hash(nombre, edad);
    }

    @Override
    public String toString() {
        return "Persona{" +
                "nombre='" + nombre + '\'' +
                ", edad=" + edad +
                '}';
    }
}
```