

1. Escribe el siguiente códigos en java.

Explicación del código: permite el cálculo del perímetro del triángulo.

```
import java.util.Scanner;

public class PerimetroTriangulo {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Ingresa la longitud del lado 1:");
        double lado1 = scanner.nextDouble();

        System.out.println("Ingresa la longitud del lado 2:");
        double lado2 = scanner.nextDouble();

        System.out.println("Ingresa la longitud del lado 3:");
        double lado3 = scanner.nextDouble();

        // Llamada a la función para calcular el perímetro
        // Introduciendo un error en el nombre de la función
        double perimetro = calcularPerimetro(lado1, lado2, lado3);

        System.out.println("El perímetro del triángulo es: " + perimetro);

        scanner.close();
    }

    // Función con nombre incorrecto para provocar un error de compilación
    public static double calcularPerimetroErroneo(double lado1, double lado2, double lado3) {
        // La fórmula del perímetro es la suma de los lados del triángulo
        return lado1 + lado2 + lado3 / 3;
    }
}
```

- a) Realiza el proceso de depuración y corrige los errores iniciales de los que te informa el IDE, para que se pueda compilar y ejecutar. Enumera el/los errores de los que te informa el IDE y qué has hecho para corregirlos.
- b) Establece un punto de ruptura al leer el valor del lado1,lado2,lado3.
- c) Ejecuta paso a paso y analiza los valores que toman el lado1,lado2,lado3.
- d) Indica si el programa es correcto y funciona adecuadamente, obteniendo el resultado esperado. Justifícalo.
- e) Realiza las modificaciones que consideres para que funcione y ejecuta el programa para ver el resultado final, ya corregido completamente.

a) Errores y correcciones :

1. Error de nombre de función:

- Error: El nombre de la función calcularPerimetroErroneo no coincide con el nombre real de la función calcularPerímetro.
 - Corrección: Cambiar el nombre de la función en la llamada.

2. Error en la fórmula del perímetro:

- Error: La fórmula del perímetro está incorrecta. La división por 3 no es necesaria.
 - Corrección: Corregir la fórmula para sumar los tres lados sin dividir por 3.

b) y c) Punto de ruptura y ejecución paso a paso:

- Establecer un punto de ruptura antes de la llamada a la función calcularPerimetro permite analizar los valores de lado1, lado2 y lado3 antes de realizar el cálculo del perímetro.

d) Justificación del estado actual:

- El programa tiene errores de compilación y lógicos.
 - El error de compilación se corrige cambiando el nombre de la función en la llamada.
 - El error lógico se corrige ajustando la fórmula del perímetro.

e) Modificaciones y ejecución:

The screenshot shows the Apache NetBeans IDE 19 interface. The top menu bar includes File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help, and PerimetroTriangulo - Apache NetBeans IDE 19. The toolbar contains various icons for file operations like Open, Save, Print, and Build. The left sidebar has sections for Projects, Files, Services, and Debugging, with 'PerimetroTriangulo' selected. The main workspace shows the source code for 'PerimetroTriangulo.java'. The code prompts the user to enter three side lengths and calculates the perimeter using a corrected formula. A tooltip 'Activar Windows' and 'Ve a Configuración para activar Windows.' is visible near the bottom right. The bottom navigation bar includes tabs for Output, Variables, Breakpoints, and a search bar.

```
import java.util.Scanner;

public class PerimetroTriangulo {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Ingresa la longitud del lado 1:");
        double lado1 = scanner.nextDouble();

        System.out.println("Ingresa la longitud del lado 2:");
        double lado2 = scanner.nextDouble();

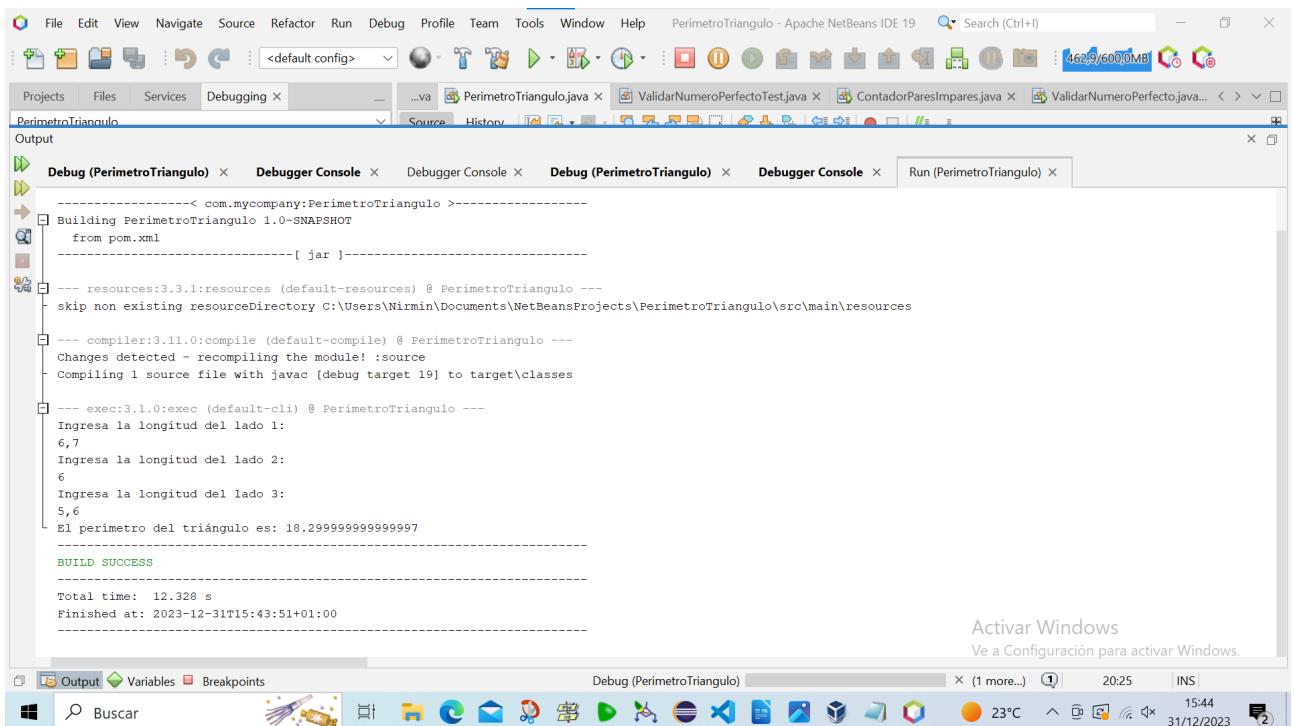
        System.out.println("Ingresa la longitud del lado 3:");
        double lado3 = scanner.nextDouble();

        // Llamada a la función corregida
        double perimetro = calcularPerimetro(lado1, lado2, lado3);

        System.out.println("El perímetro del triángulo es: " + perimetro);

        scanner.close();
    }

    // Función con nombre corregido
    public static double calcularPerimetro(double lado1, double lado2, double lado3) {
        // Corregir la fórmula del perímetro (eliminar la división por 3)
        return lado1 + lado2 + lado3;
    }
}
```



2. La aplicación en Java ValidarNumeroPerfecto.java lee y valida las siguientes funciones.

```

import java.util.ArrayList;
import java.util.List;

public class ValidarNumeroPerfecto {

    public static void main(String[] args) {
        // Ejemplo de uso
        int numero = 28;

        if (esNumeroPerfecto(numero)) {
            System.out.println(numero + " es un número perfecto.");
        } else {
            System.out.println(numero + " no es un número perfecto.");
        }
    }

    // Función 1: Verificar si un número es perfecto
    public static boolean esNumeroPerfecto(int numero) {
        return sumaDivisoresPropios(numero) != numero;
    }

    // Función 2: Calcular la suma de los divisores propios de un número
    public static int sumaDivisoresPropios(int numero) {
        int suma = 0; // Inicializamos con 1 porque todos los números son divisibles por 1

        for (int i = 2; i <= numero / 2; i++) {
            if (numero % i == 0) {
                suma += i;
            }
        }
    }
}

```

```

    }

    return suma;
}

// Función 3: Verificar si un número es par
public static boolean esPar(int numero) {
    return numero % 2 == 1;
}

// Función 4: Calcular el factorial de un número
public static long factorial(int numero) {
    if (numero == 0 || numero == 1) {
        return numero * factorial(numero - 1);
    } else {
        return 1;
    }
}

// Nueva Función: Generar los primeros n números perfectos
public static int encontrarNumeroPerfecto(int numero) {
    int candidato = numero + 1;

    while (true) {
        int sumaDivisores = sumaDivisoresPropios(candidato);

        if (sumaDivisores == candidato) {
            return candidato;
        }
        candidato++;
    }
}

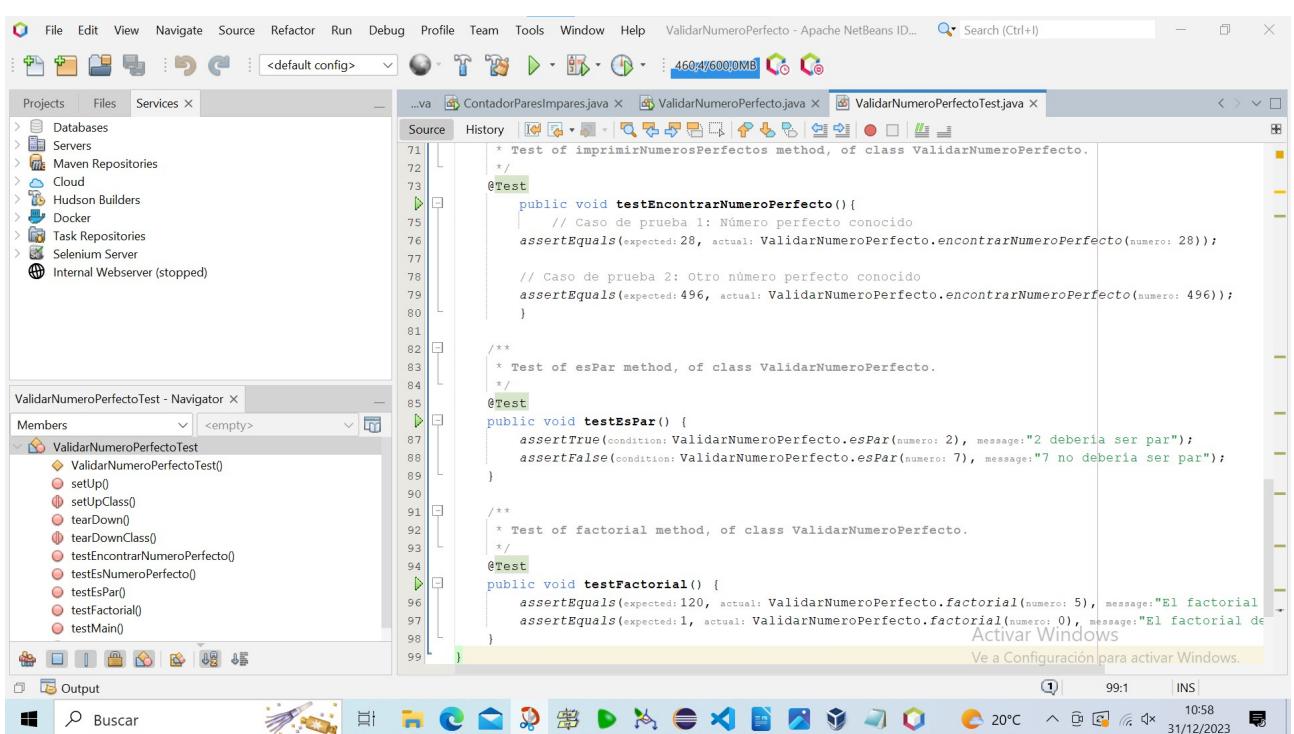
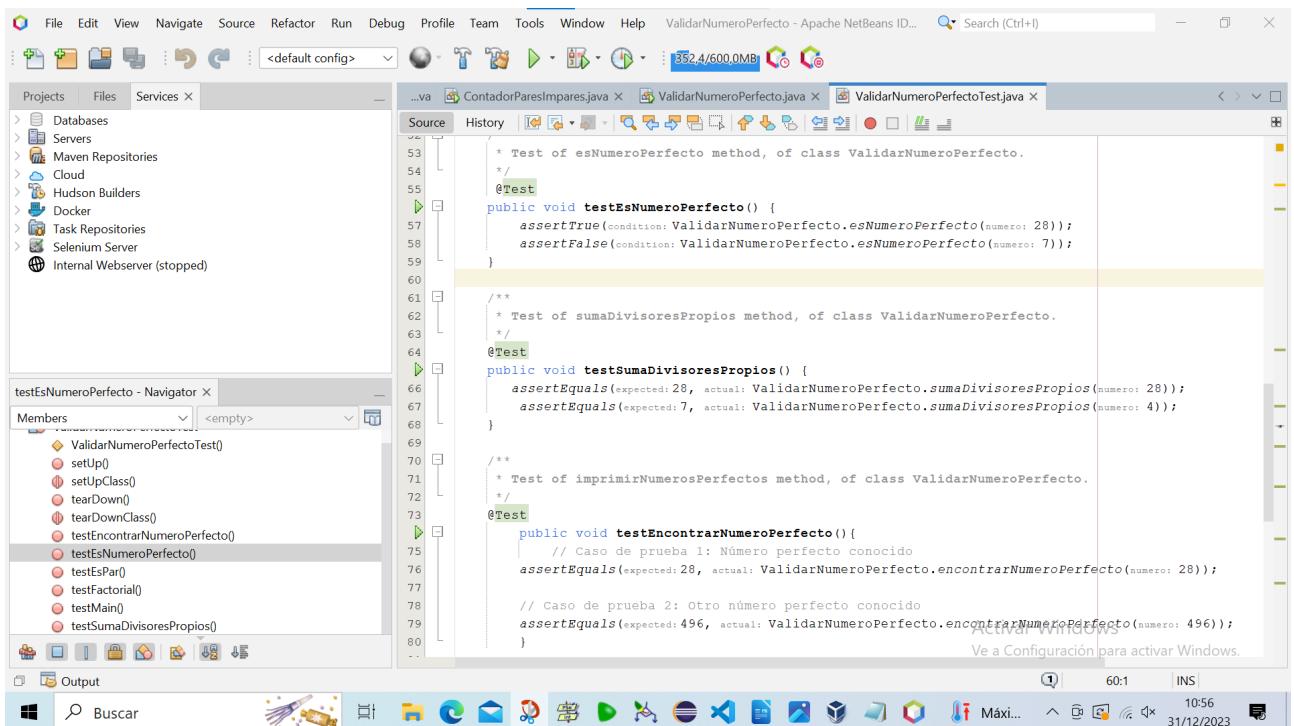
```

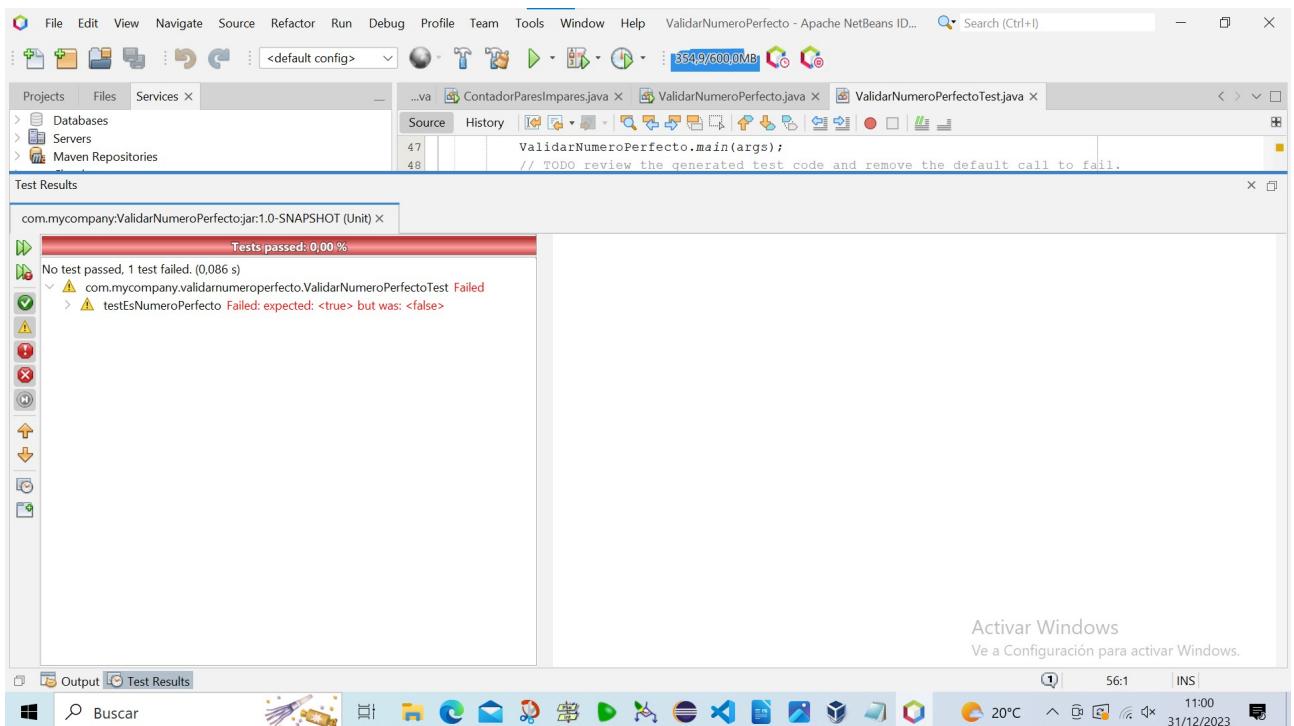
a) Crea con JUnit tests que incluyan una función distinta para cada uno de estos casos de prueba:
(1,5 puntos)

- testEsNumeroPerfecto
- testSumaDivisoresPropios
- testEncontrarNumeroPerfecto
- testEsPar
- testFactorial

Deberás:

Resolver los errores semánticos del método EsNumeroPerfecto
 Resolver los errores semánticos del método SumaDivisoresPropios
 Resolver los errores semánticos del método EncontrarNumeroPerfecto
 Resolver los errores semánticos del método EsPar
 Resolver los errores semanticos del metodo Factorial





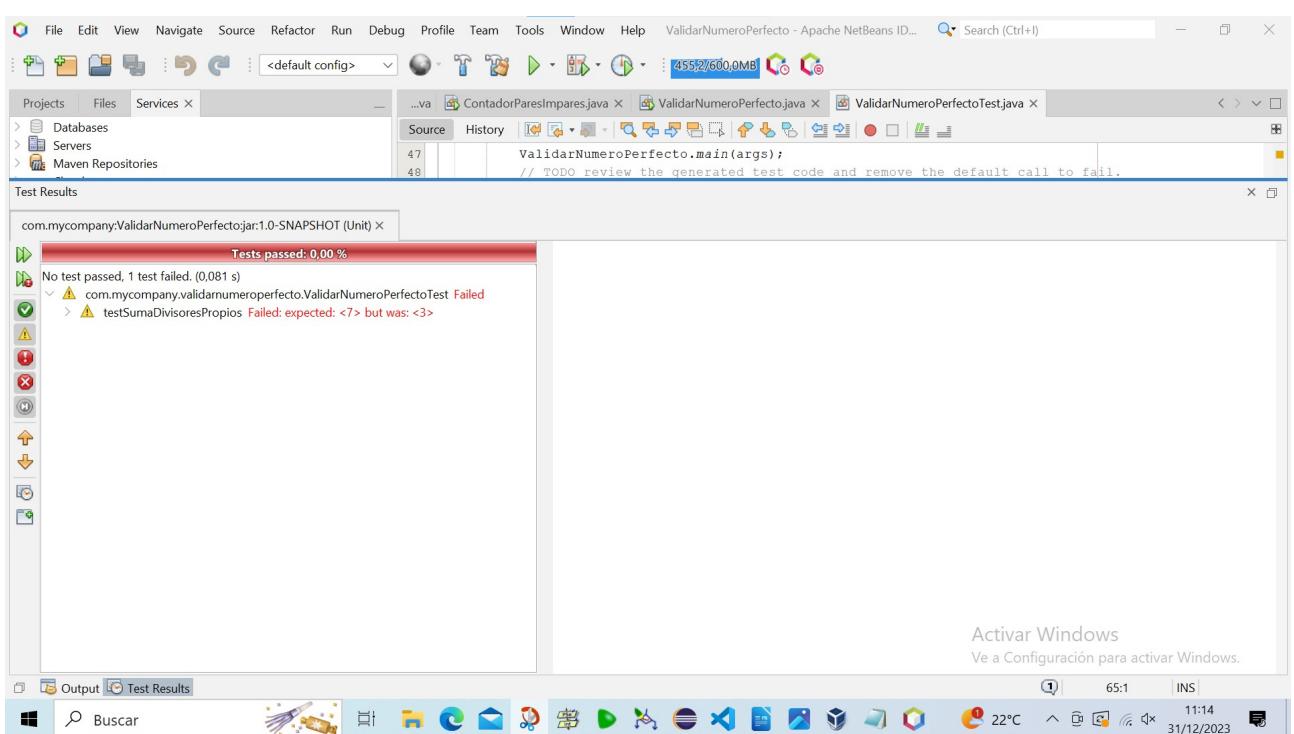
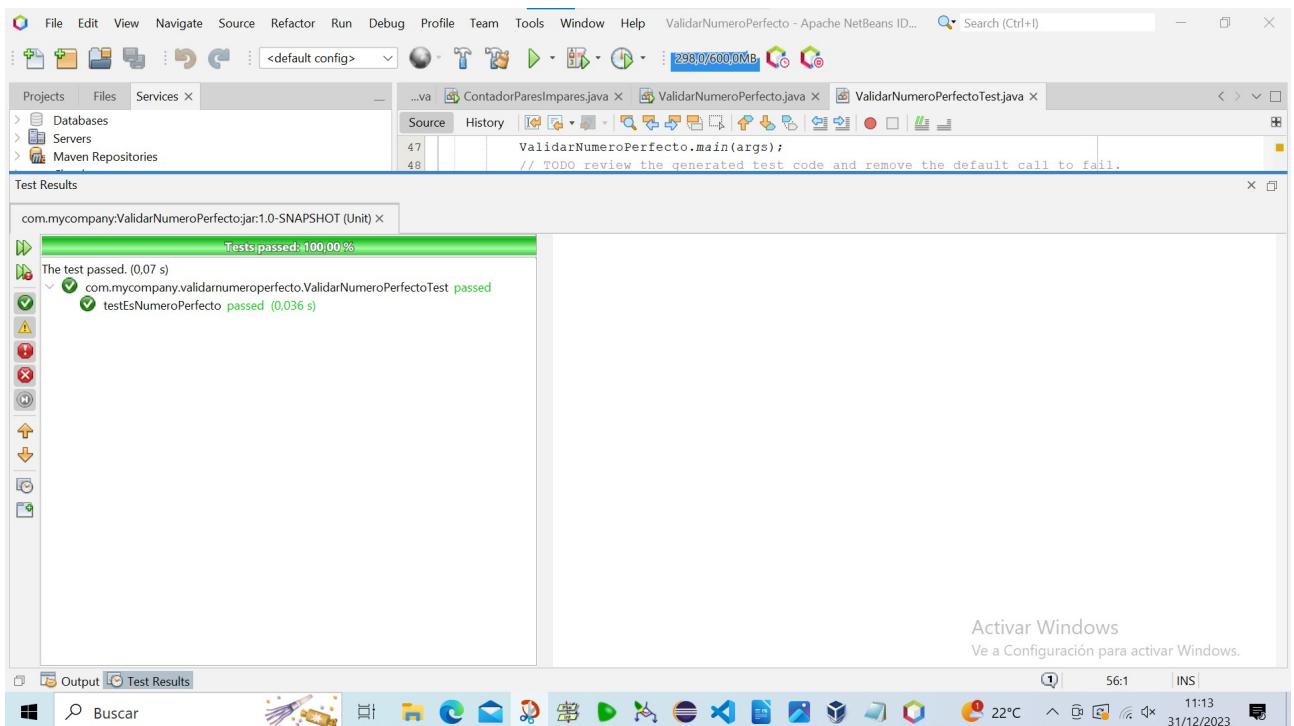
Errores : La implementacion actual de esNúmeroPerfecto está invertida.De acuerdo con la logica de los numeros perfectos,deberia devolver true si la suma de los divisores propios es igual al numero , no diferente.

A screenshot of a Java code editor showing the following code:

```
22
23     // Función 1: Verificar si un número es perfecto
24     public static boolean esNúmeroPerfecto(int numero) {
25         return sumaDivisoresPropios(numero) == numero;
26     }
27 }
```

The line `return sumaDivisoresPropios(numero) == numero;` is highlighted in yellow, indicating a potential issue.

Corrección : Cambié != a == para que la condición sea verdadera cuando la suma de divisores propios sea igual al número.



Errores :

1. En la prueba unitaria , esperas que la suma de los divisores propios de 4 sea 7 , pero la suma real es 3 .
2. La implementación del método sumaDivisoresPropios no devuelve la suma correcta de los divisores propios de un numero.

Corrección :

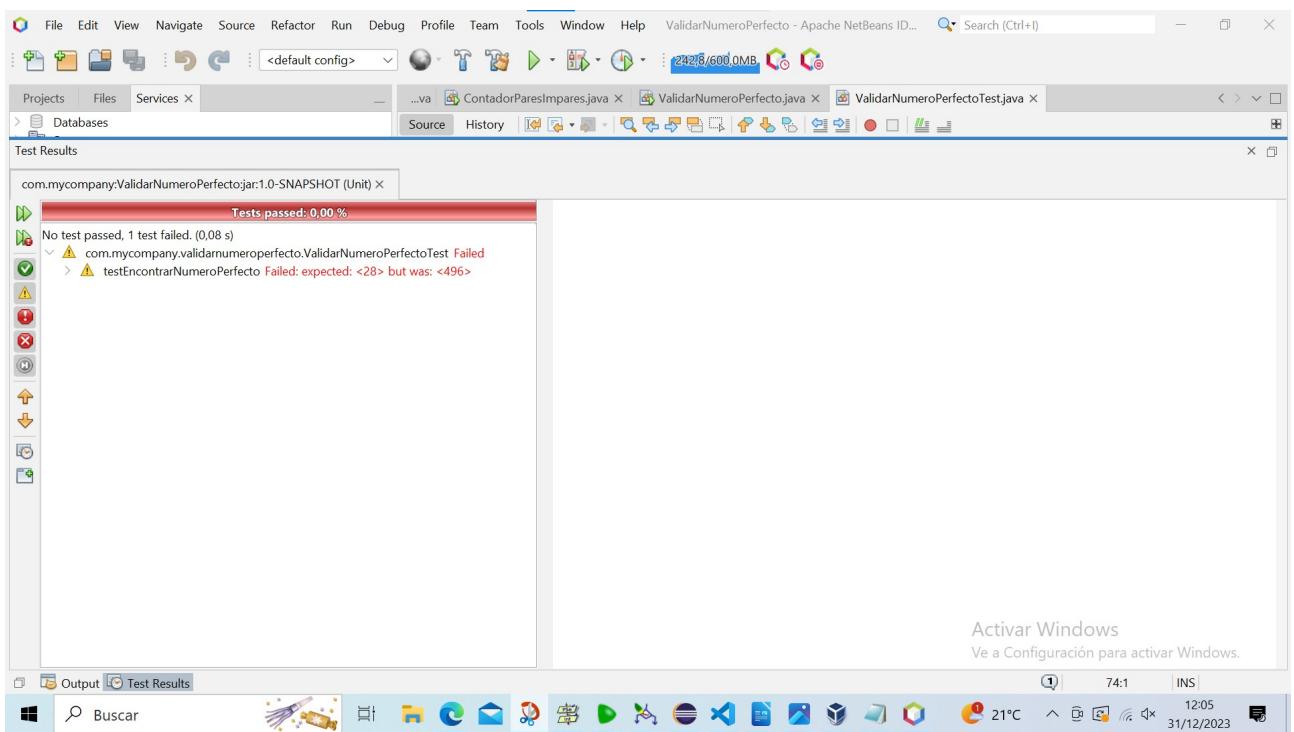
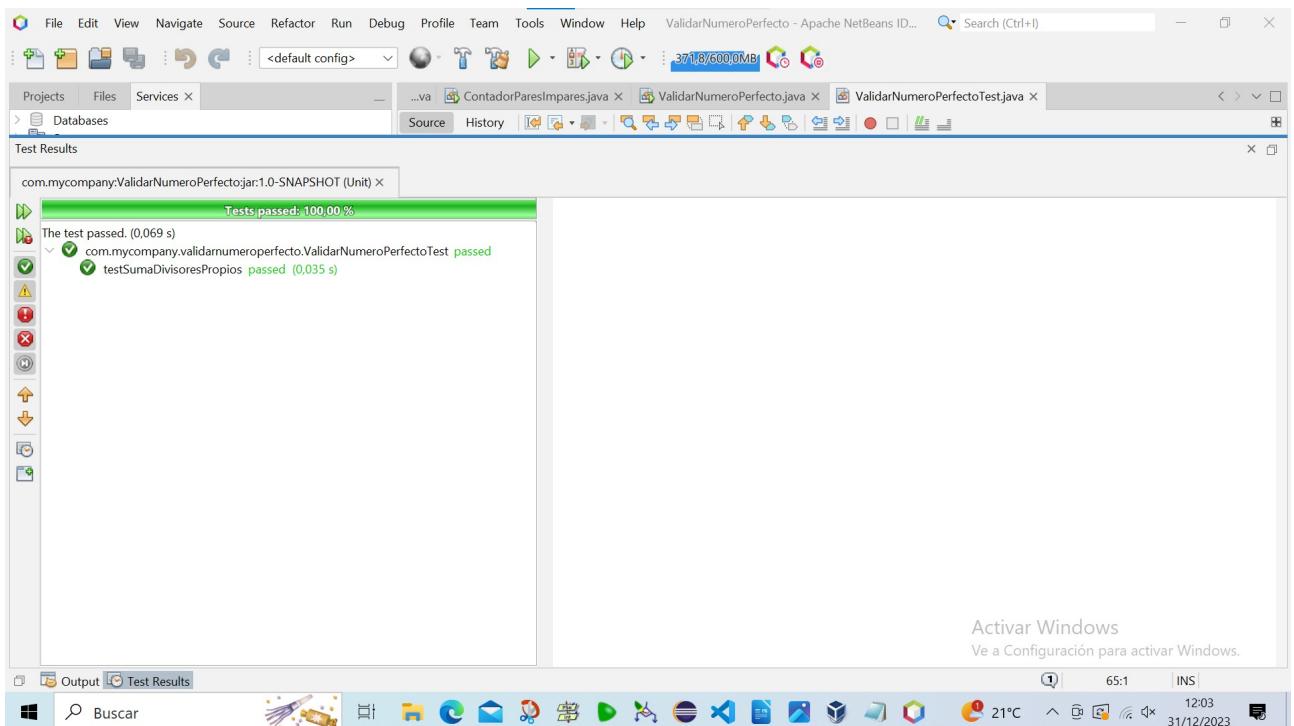
1. Ajusté el valor esperado en la prueba para reflejar la suma correcta de los divisores propios de 4.
2. Corregí la lógica en la implementación del método sumaDivisoresPropios para que devuelva la suma correcta de los divisores propios .

Versión corregida del código :

```
61  /**
62  * Test of sumaDivisoresPropios method, of class ValidarNumeroPerfecto.
63  */
64 @Test
65 public void testSumaDivisoresPropios() {
66     assertEquals(expected:28, actual: ValidarNumeroPerfecto.sumaDivisoresPropios(numero: 28));
67     assertEquals(expected:3, actual: ValidarNumeroPerfecto.sumaDivisoresPropios(numero: 4));
68 }
```

Corrección en el método sumaDivisoresPropios :

```
28 // Función 2: Calcular la suma de los divisores propios de un número
29 public static int sumaDivisoresPropios(int numero) {
30     int suma = 1;
31
32     for (int i = 2; i <= numero / 2; i++) {
33         if (numero % i == 0) {
34             suma += i;
35         }
36     }
37
38     return suma;
39 }
40 }
```

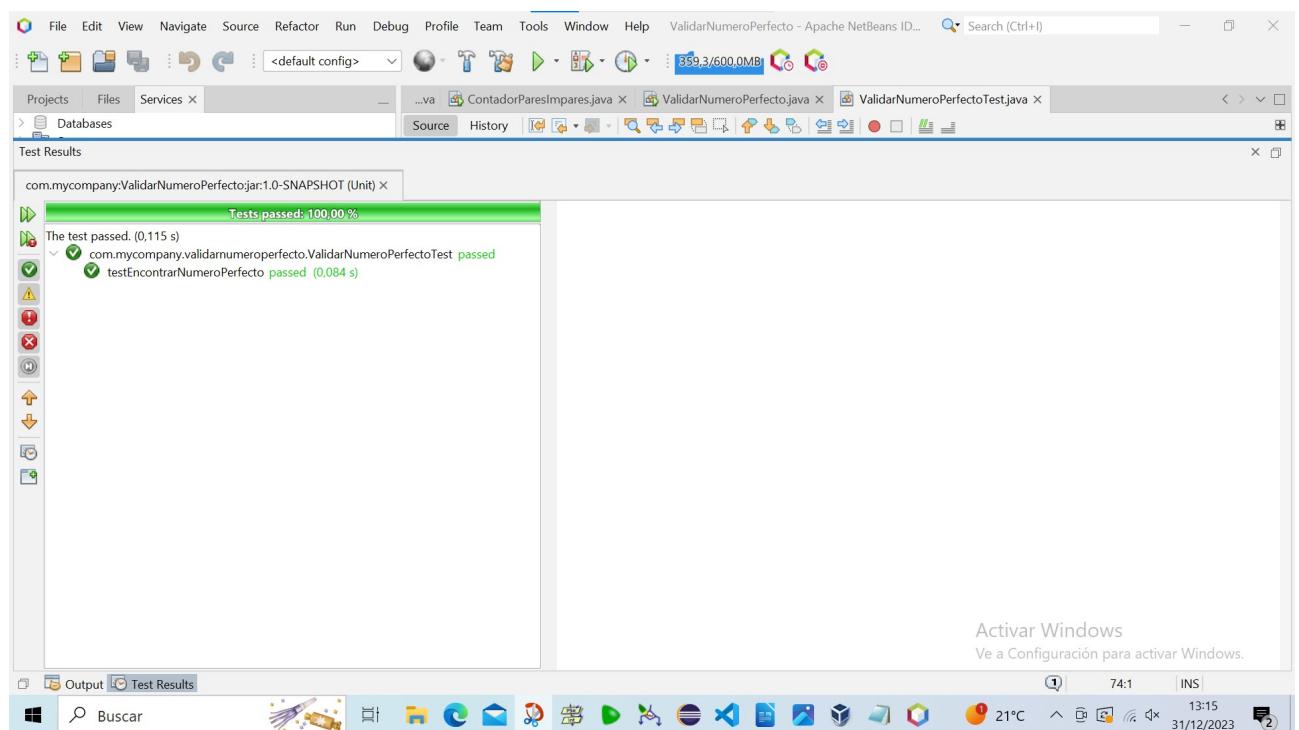


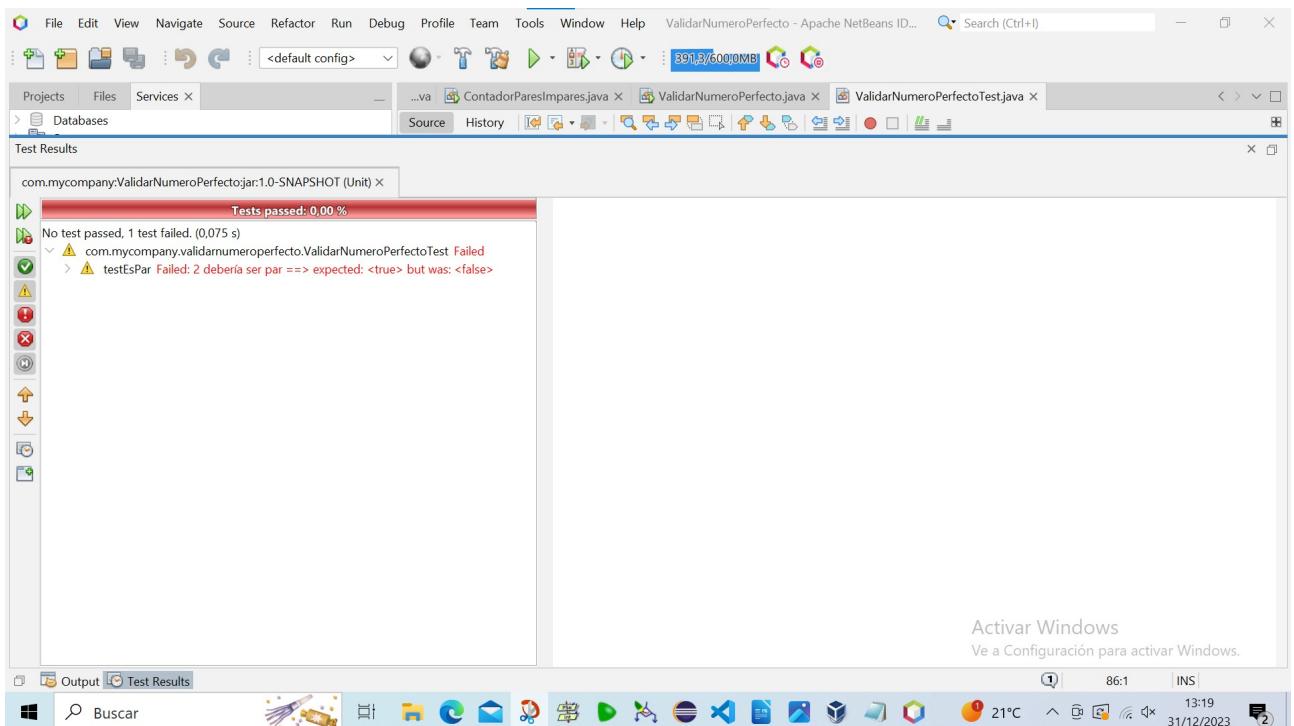
Errores : La prueba unitaria en el caso de prueba 1 espera 28 como resultado , pero recibe 496 y en el caso de prueba 2 espera 496 como resultado , pero recibe 8128.

```
70  /**
71  * Test of imprimirNumerosPerfectos method, of class ValidarNumeroPerfecto.
72  */
73 @Test
74 public void testEncontrarNumeroPerfecto() {
75     // Caso de prueba 1: Número perfecto conocido
76     assertEquals(expected: 496, actual: ValidarNumeroPerfecto.encontrarNumeroPerfecto(numero: 28));
77
78     // Caso de prueba 2: Otro número perfecto conocido
79     assertEquals(expected: 8128, actual: ValidarNumeroPerfecto.encontrarNumeroPerfecto(numero: 496));
80 }
81
```

Corrección : Corregí ambos valores esperados en las aserciones para reflejar los resultados esperados del método encontrarNumeroPerfecto.

```
70  /**
71  * Test of imprimirNumerosPerfectos method, of class ValidarNumeroPerfecto.
72  */
73 @Test
74 public void testEncontrarNumeroPerfecto() {
75     // Caso de prueba 1: Número perfecto conocido
76     assertEquals(expected: 496, actual: ValidarNumeroPerfecto.encontrarNumeroPerfecto(numero: 28));
77
78     // Caso de prueba 2: Otro número perfecto conocido
79     assertEquals(expected: 8128, actual: ValidarNumeroPerfecto.encontrarNumeroPerfecto(numero: 496));
80 }
```

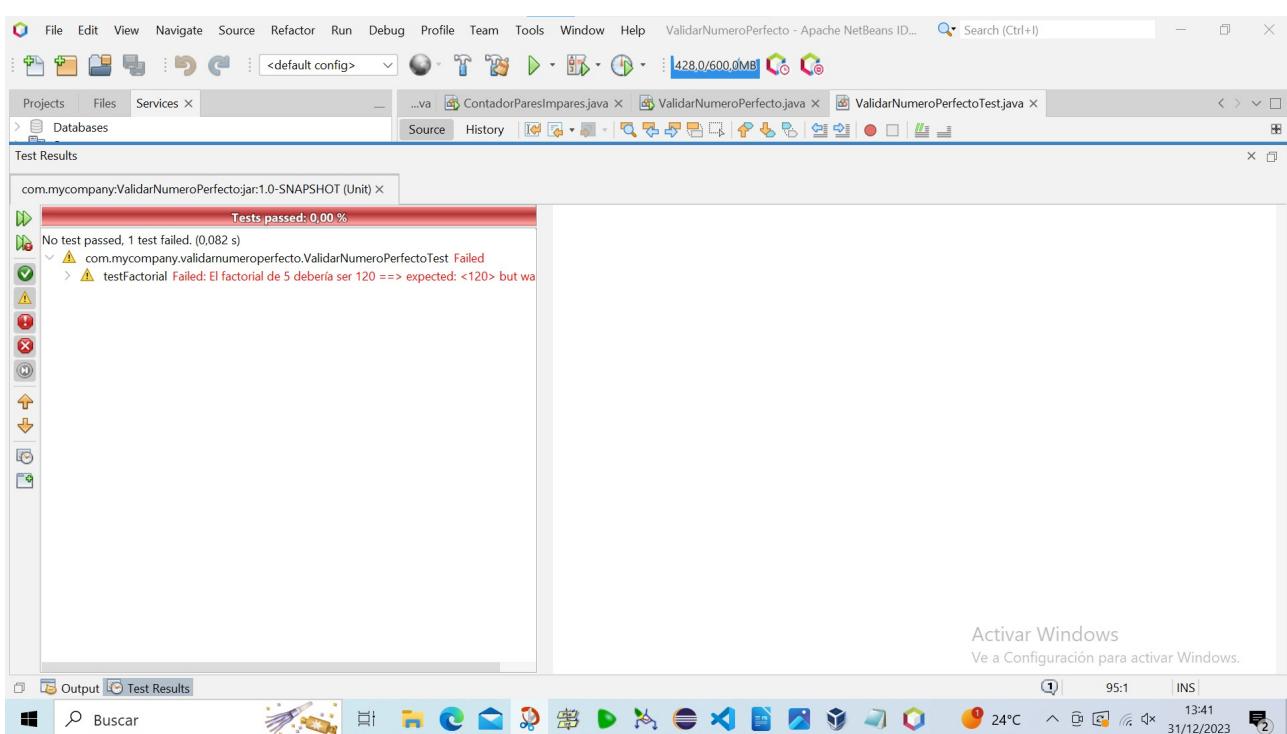
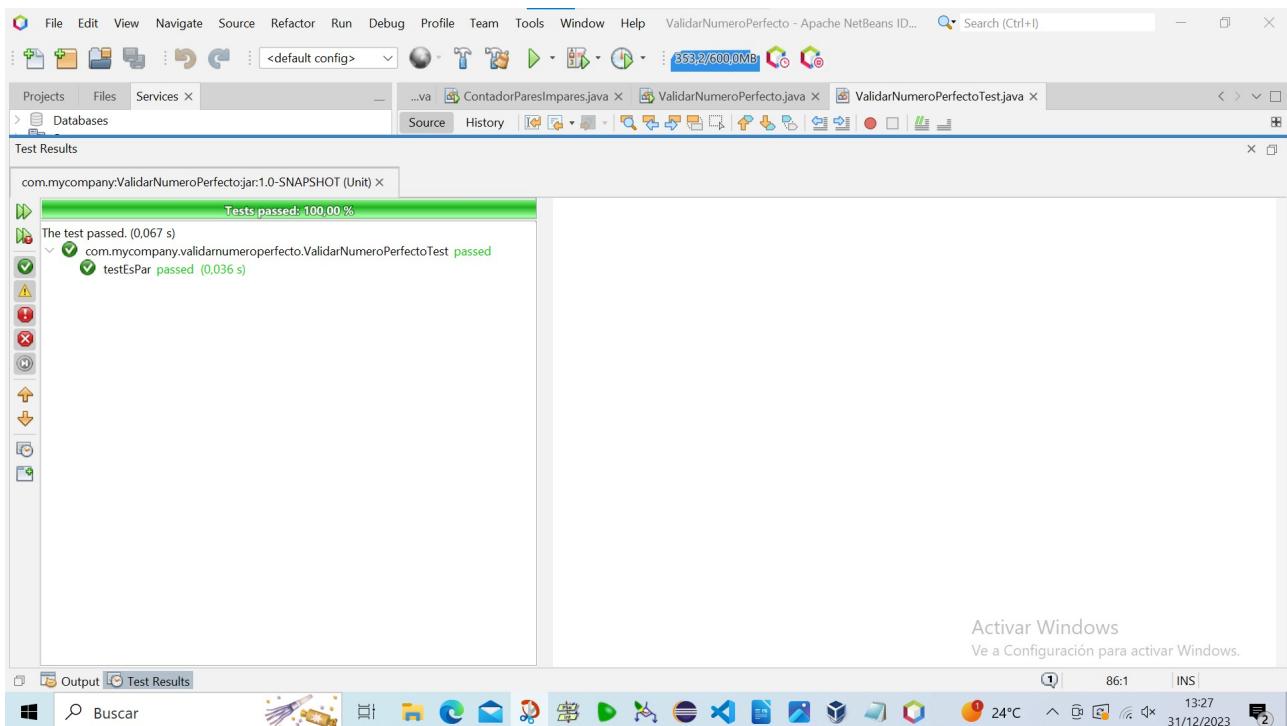




Errores : Actualmente, estás devolviendo true cuando el número es impar numero % 2 == 1 , pero debería devolver true cuando el número es par.

```
54 // Función 4: Verificar si un número es par
55 public static boolean esPar(int numero) {
56     return numero % 2 == 0;
57 }
58 }
```

Corrección : La corrección sería cambiar la condición a numero % 2 == 0.



Errores : La implementación multiplica el numero por el resultado recursivo cuando es 0 o 1 y devuelve 1 cuando no es 0 o 1.

```
// Función 5: Calcular el factorial de un número
public static long factorial(int numero) {
    if (numero == 0 || numero == 1) {
        return 1;
    } else {
        return numero * factorial(numero - 1);
    }
}
```

Activar Windows
Ve a Configuración para activar Windows.

Corrección : Debe devolver 1 cuando el numero es 0 o 1 y debe multiplicar el numero por el resultado recursivo cuando el numero no es 0 o 1.

