Seguimiento

Programación 1

Jader Alexis Riascos - Maria Luisa Londoño - Juan Manuel Orozco

Ejercicio #1.

**¿Qué es una máquina virtual?**

Una máquina virtual a veces se percibe como "una computadora dentro de una computadora" o "una computadora simulada en software". Sin embargo, esta intuición puede ser engañosa.

**¿Qué es el bytecode?**

En el contexto de Java, el bytecode de la JVM no es muy similar al código de máquina real del hardware. Se considera una representación intermedia entre el código fuente y el código de máquina.

**¿Es javac un compilador?**

Javac produce bytecode, no código de máquina. En términos de ciencia de la computación, javac es más similar a la mitad frontal de un compilador, creando una representación intermedia.

**¿Por qué se llama "bytecode"?**

Se llama "bytecode" porque el código de instrucción (opcode) es solo un solo byte. Hay alrededor de 256 instrucciones posibles, y algunas de ellas no se utilizan.

**¿El bytecode está optimizado?**

En los primeros días, Javac producía bytecode altamente optimizado, pero con la llegada de la compilación JIT, se considera un error optimizar demasiado el bytecode.

**¿El bytecode realmente es independiente de la máquina? ¿Y qué hay de cosas como el endianness?**

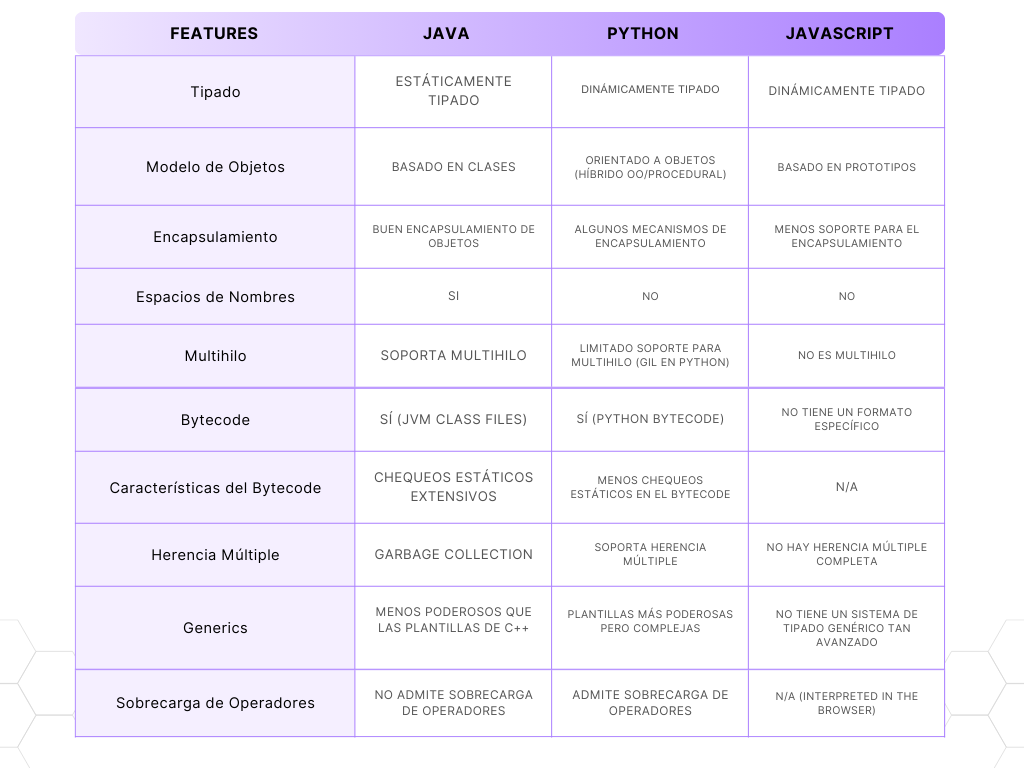
Sí, el formato del bytecode siempre es el mismo, independientemente de la máquina en la que se creó. Esto incluye el orden de bytes (endianness), que siempre es big-endian.

**¿Java es un lenguaje interpretado?**

La JVM es esencialmente un intérprete con compilación JIT para el rendimiento. Java sigue un proceso de compilación, pero la distinción entre lenguajes "interpretados" y "compilados" se está volviendo borrosa en otros entornos de programación.

**¿Otras lenguas pueden ejecutarse en la JVM?**

Sí, la JVM puede ejecutar cualquier archivo de clase válido, lo que permite que lenguajes no Java se ejecuten en ella. Esto se puede lograr a través de compiladores de código fuente (por ejemplo, Kotlin, Scala) o mediante la implementación de intérpretes y entornos de ejecución en Java (por ejemplo, JRuby).

B)

C)

**Comentarios en Java:**

Los comentarios en Java son textos en lenguaje natural destinados a lectores humanos de un programa. El compilador de Java los ignora y sirven como medio para documentar el código para una mejor comprensión. Java admite tres tipos de comentarios.

**Comentario de una sola línea:**

Comienza con // y continúa hasta el final de la línea actual.

Ejemplo: int i = 0; // Inicializa la variable del bucle

**Comentario Multilínea:**

Comienza con /\* y termina con \*/.

Se utiliza para comentarios de varias líneas, pero también se puede utilizar para comentarios de una sola línea.

Ejemplo:

/\*

\* Primero, establezca una conexión con el servidor.

\* Si el intento de conexión falla, salga de inmediato.

**Comentario del médico:**

Caso especial de comentario multilínea que comienza con /\*\*.

Se utiliza para documentar clases y métodos para la extracción automatizada mediante herramientas como Javadoc.

**Ejemplo:**

/\*\*

\* Upload a file to a web server.

\*

\* @param file The file to upload.

\* @return <tt>true</tt> on success,

\* <tt>false</tt> on failure.

\* @author David Flanagan

\*/

Los comentarios pueden aparecer entre cualquier token de un programa Java, pero no dentro de un token. No pueden aparecer dentro de literales de cadenas entre comillas dobles.

**Palabras Reservadas:**

Las palabras reservadas en Java son parte de la sintaxis del lenguaje y no se pueden usar como nombres de variables o clases. Algunos ejemplos incluyen:

Palabras clave: abstract, const, final, int, public, throw, etc.

Literales: true, false, null.

Nota: const y goto son reservadas pero no se utilizan en el lenguaje.

**Identificadores:**

Un identificador en Java es un nombre dado a alguna parte del programa, como una clase, un método o una variable. Reglas para identificadores:

Puede tener cualquier longitud y contener letras y dígitos de todo el conjunto de caracteres Unicode.

No puede comenzar con un dígito.

No puede contener caracteres de puntuación, excepto $ y símbolos de moneda Unicode.

El guión bajo (\_) estaba permitido pero ya no se permite como identificador en versiones recientes.

Ejemplos de identificadores legales: i, x1, theCurrentTime, current, 獺 (identificador UTF-8).

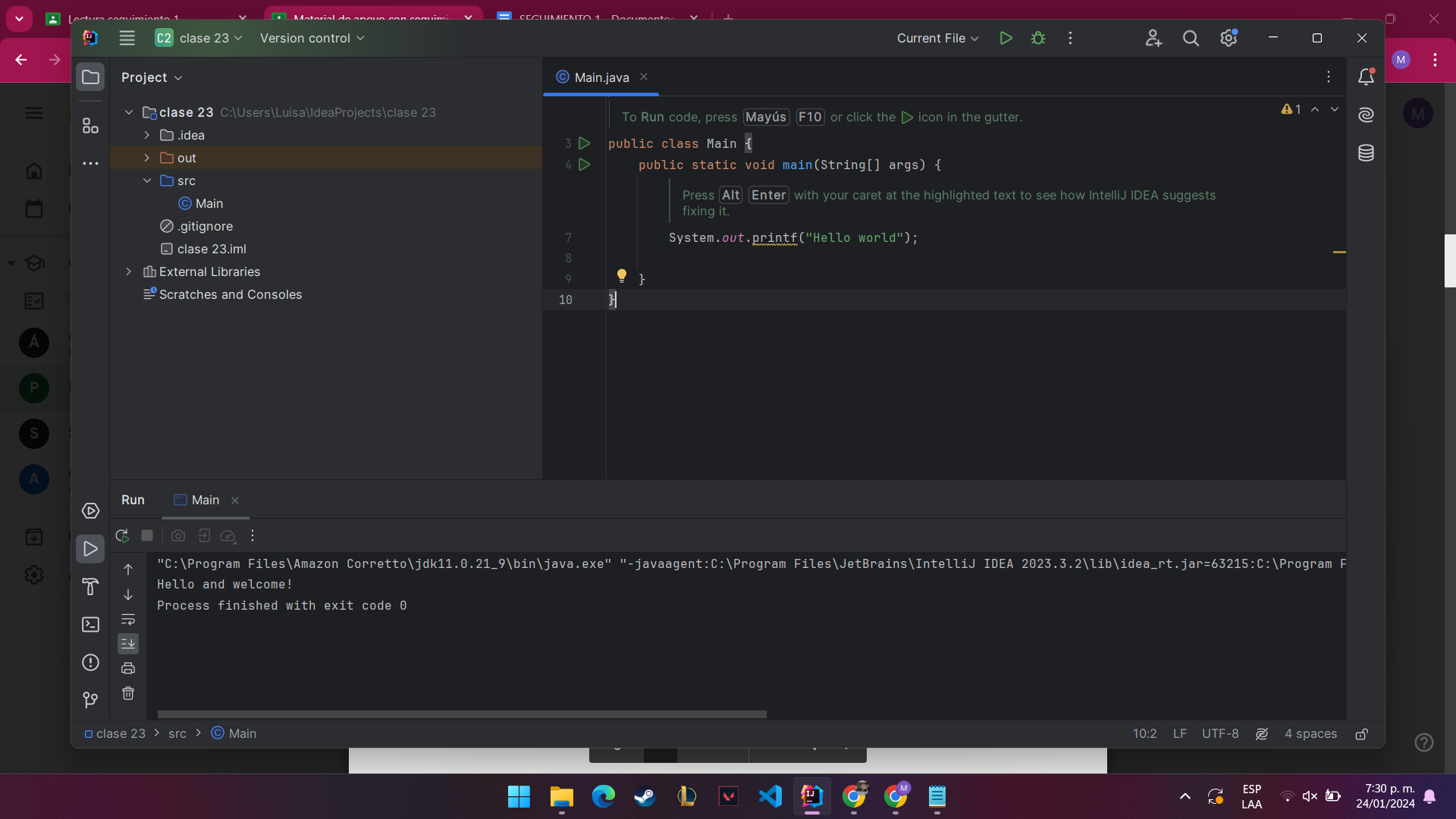
**Literales:**

Los literales en Java son secuencias de caracteres que representan valores constantes en el código fuente. Incluyen literales numéricos, de caracteres y de cadenas, así como palabras reservadas como true, false y null.

Ejemplos de literales: 1, 1.0, '1', 1L, "one", true, false, null.

Para obtener una sintaxis detallada sobre la expresión de literales numéricos, de caracteres y de cadenas, consulta la sección "Tipos de Datos Primitivos".

Ejercicio #2.

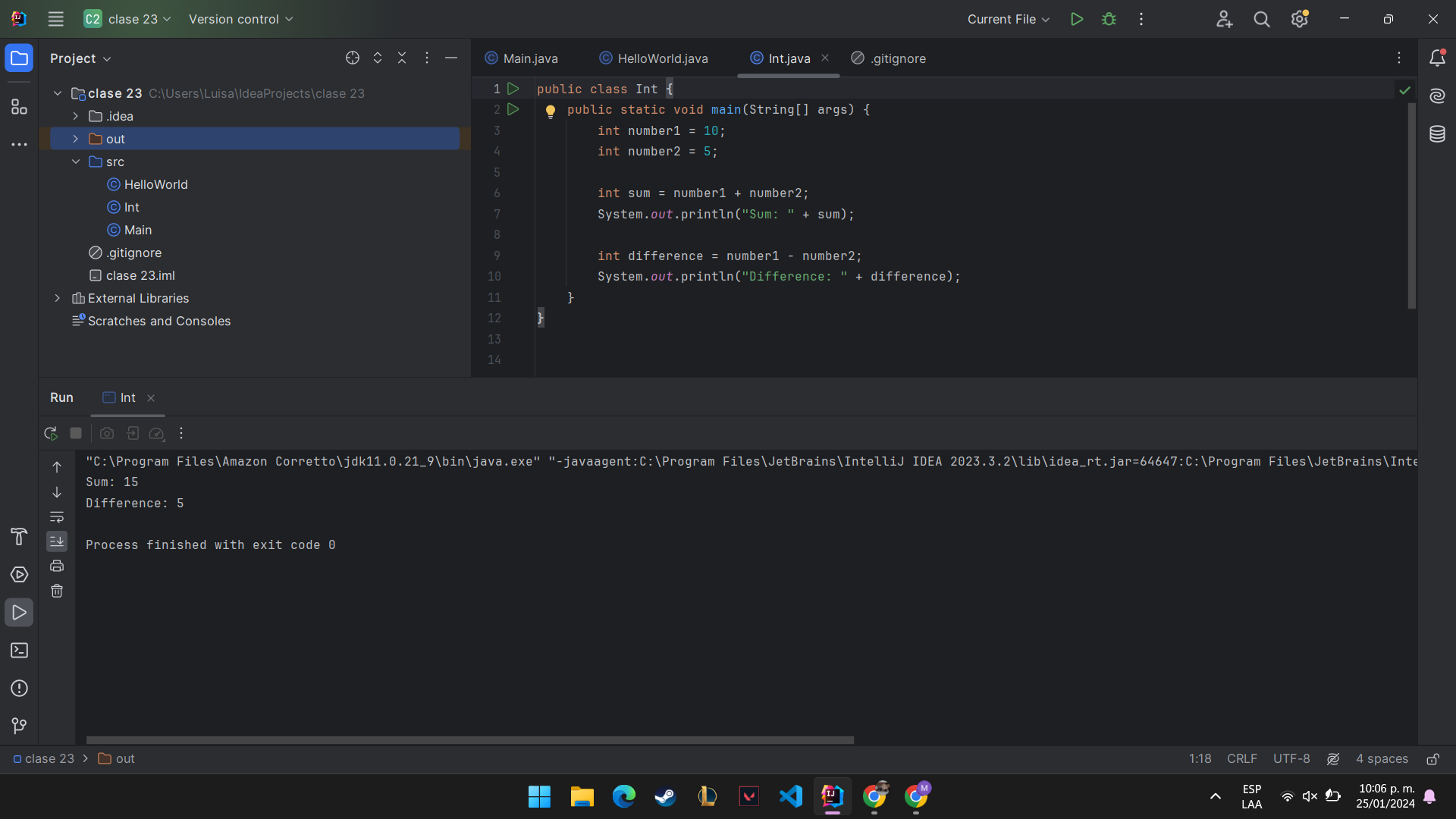


Ejercicio #3.

En Java, los tipos de datos primitivos son tipos de datos fundamentales que representan valores básicos y no son objetos. Estos tipos de datos son admitidos directamente por el lenguaje y no requieren instancia. Los tipos de datos primitivos en Java se clasifican en cuatro tipos: enteros, números de punto flotante, caracteres y booleanos. Aquí tienes ejemplos de cada tipo junto con su salida en la consola:

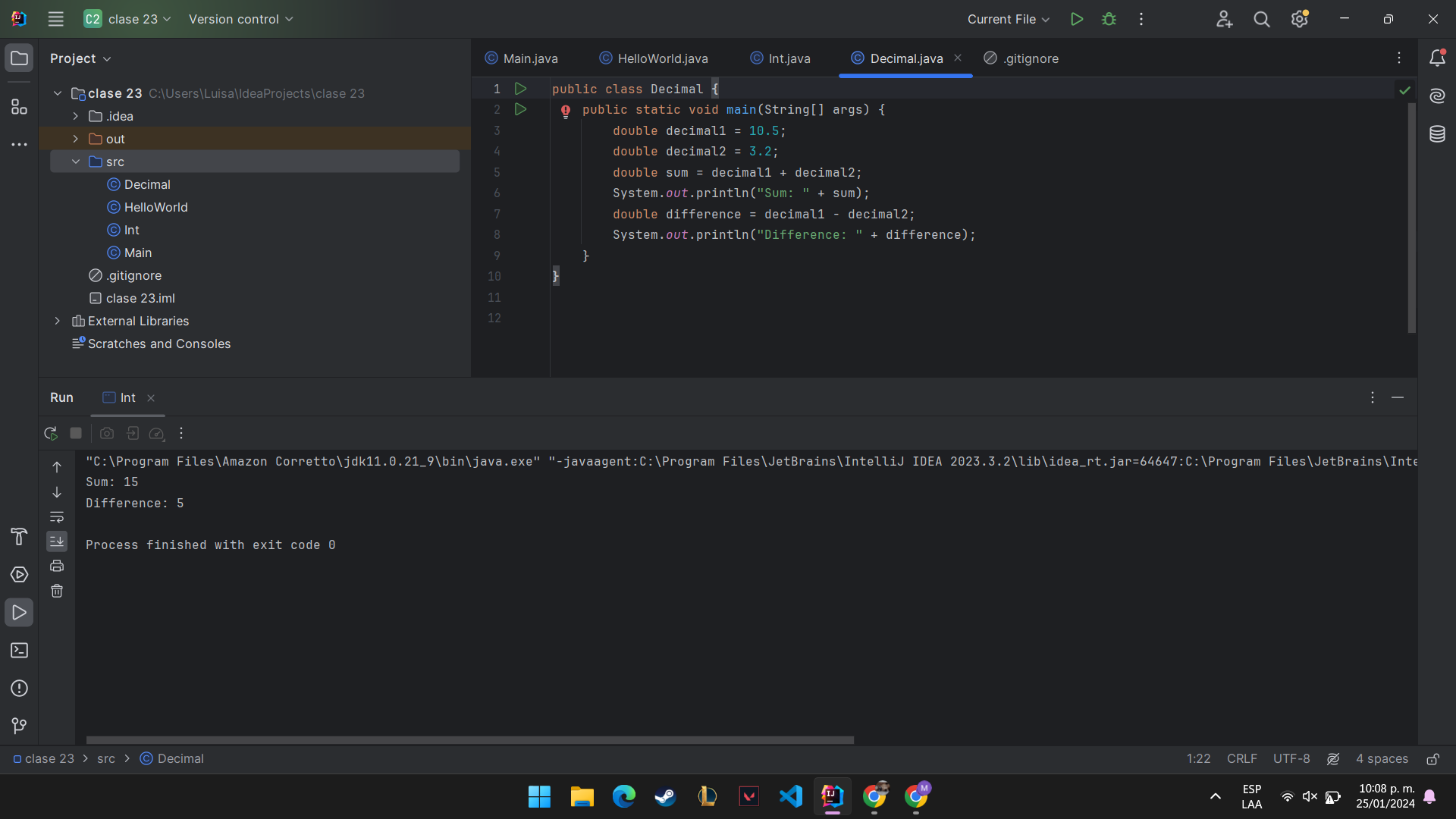
**a) Enteros:**

Tipos de datos primitivos que representan números enteros sin parte decimal.



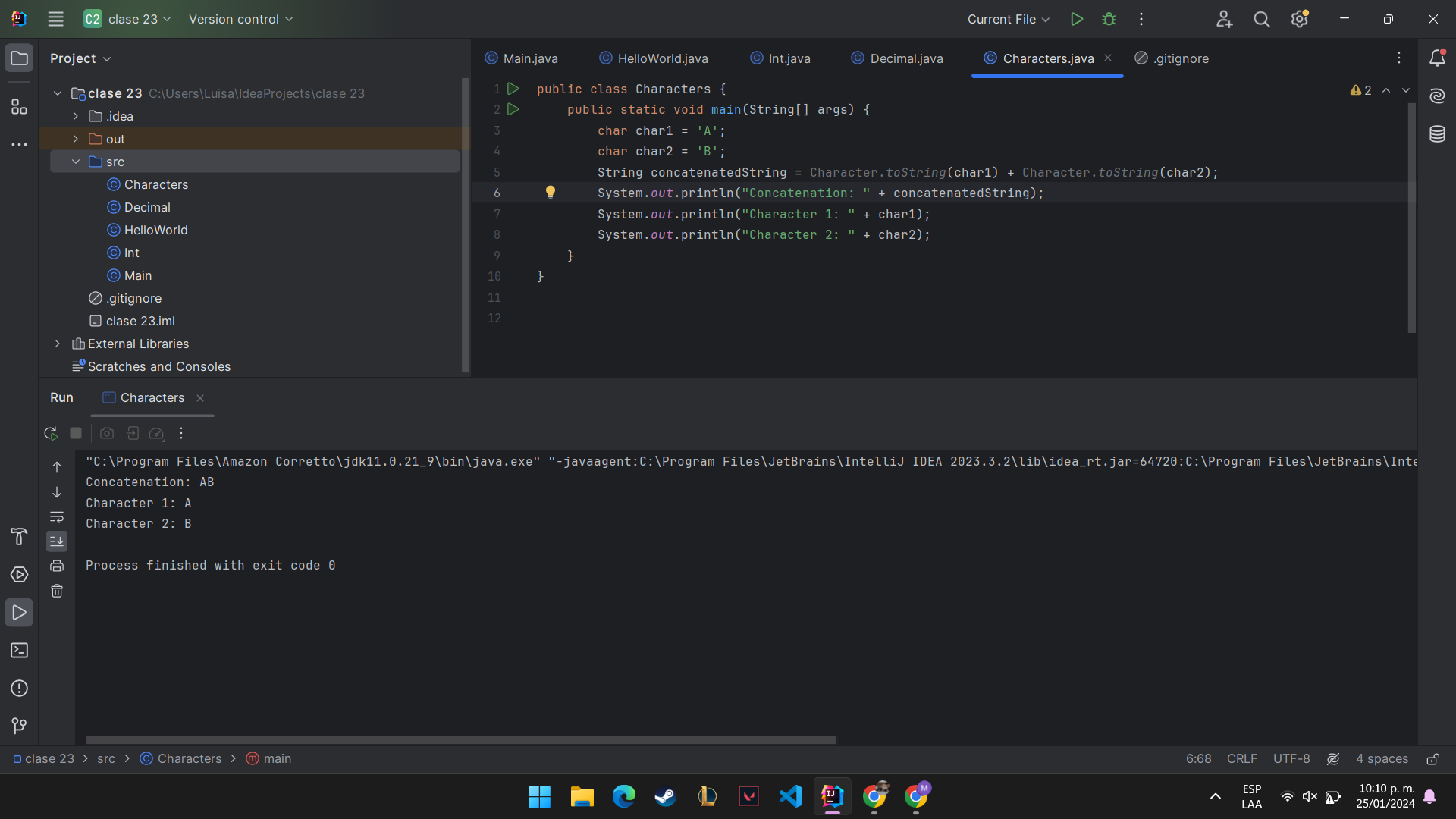
**b) Decimales**

Tipos de datos primitivos que representan números con una parte decimal.



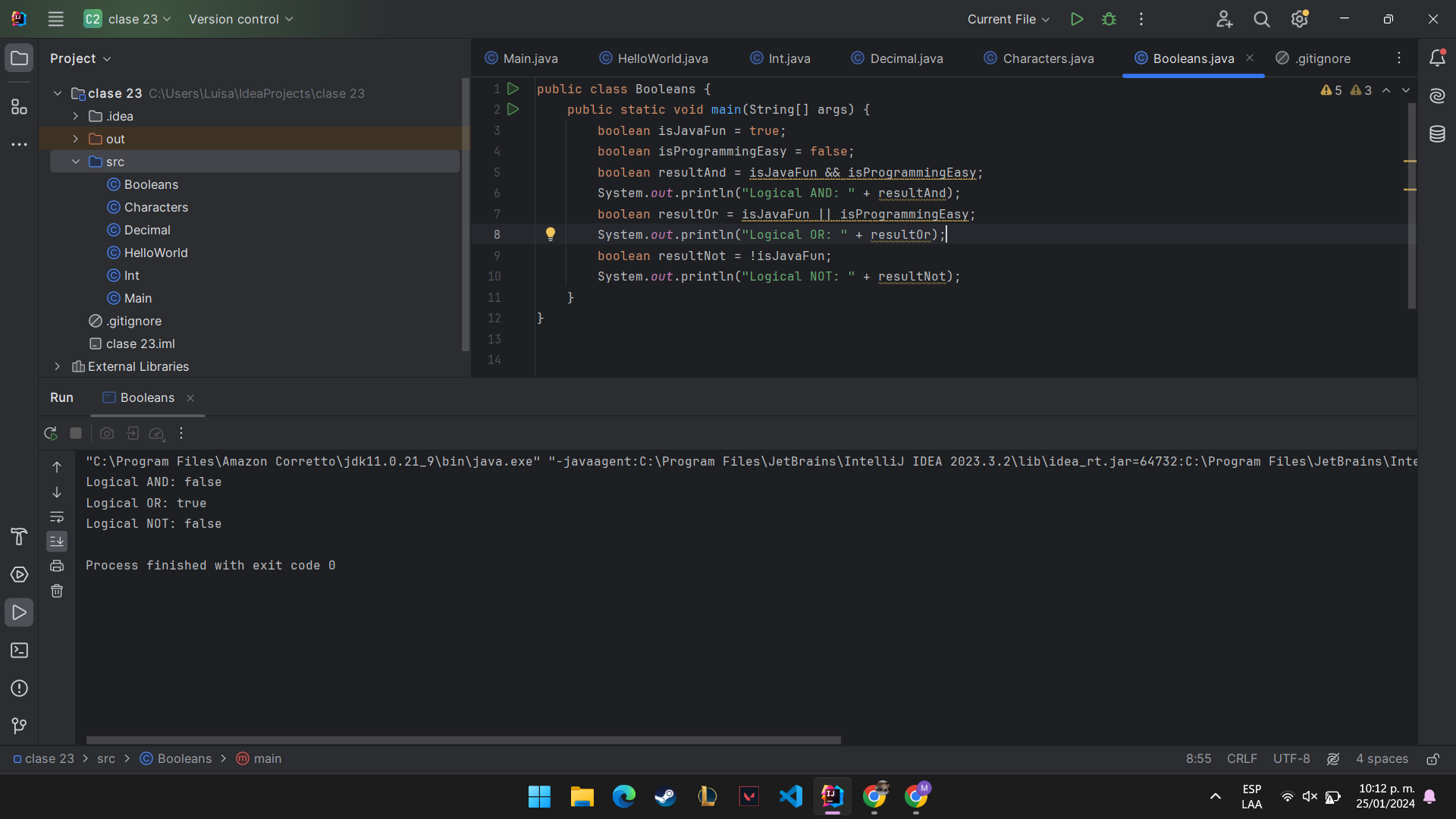
**c) Carácteres:**

Tipos de datos primitivos que representan un único carácter Unicode.



**d) Booleanos:**

Tipos de datos primitivos que representan valores de verdad (verdadero o falso).



Ejercicio #4.

**OPERADORES ARITMÉTICOS**

A. Suma (+):  


B. Resta (-):



C. Multiplicación (\*):  


**OPERADORES LÓGICOS**

A. AND (&&)



B. OR (||):  


C. NOT (!)

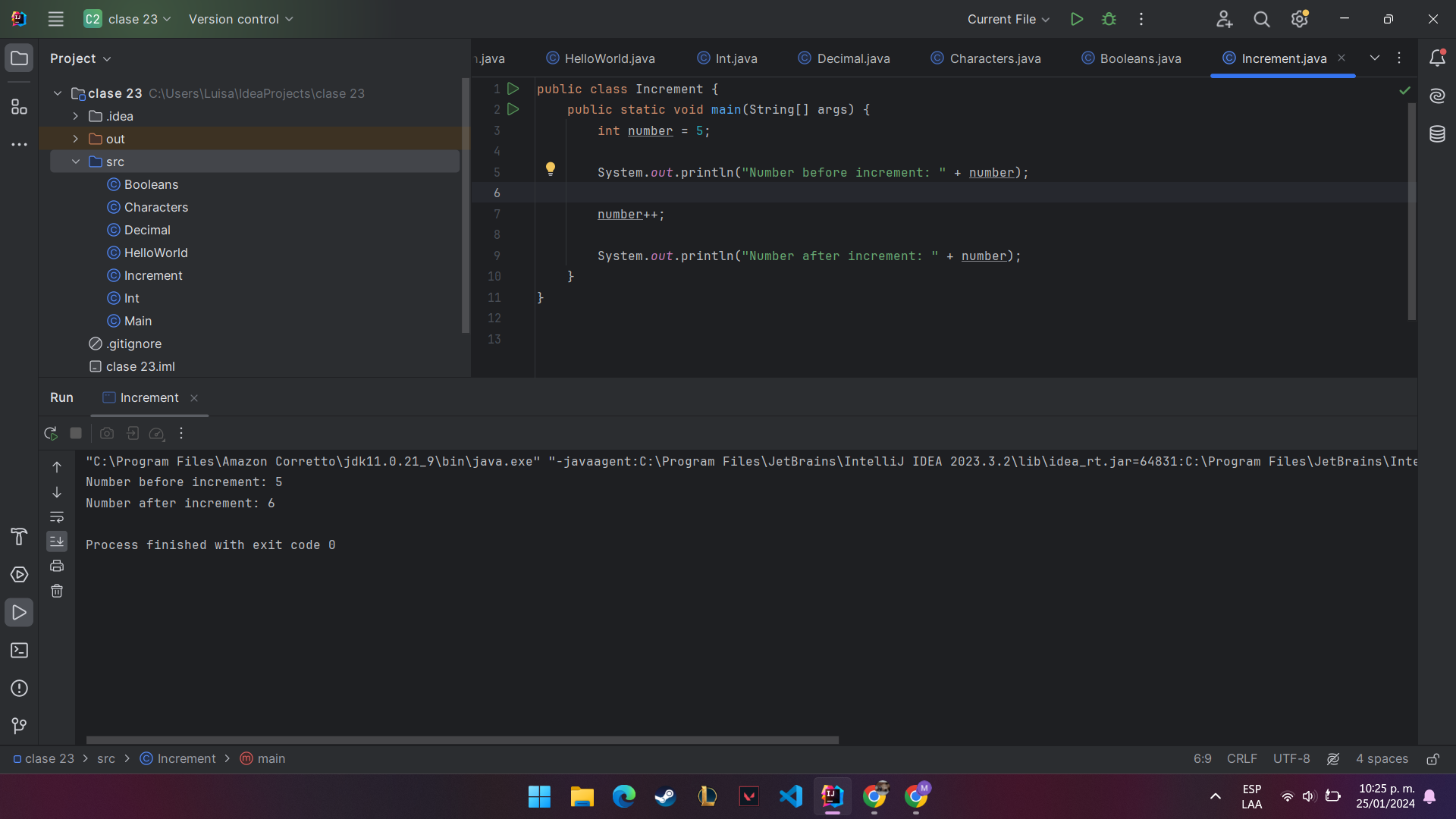


Ejercicio #5.

En Java, los operadores de incremento (++) y decremento (--) son operadores unarios utilizados para aumentar o disminuir en 1 el valor de una variable, respectivamente. Estos operadores se pueden aplicar a tipos de datos enteros (byte, short, int, long) y tipos de datos de punto flotante (float, double). Así es cómo funcionan con ejemplos:

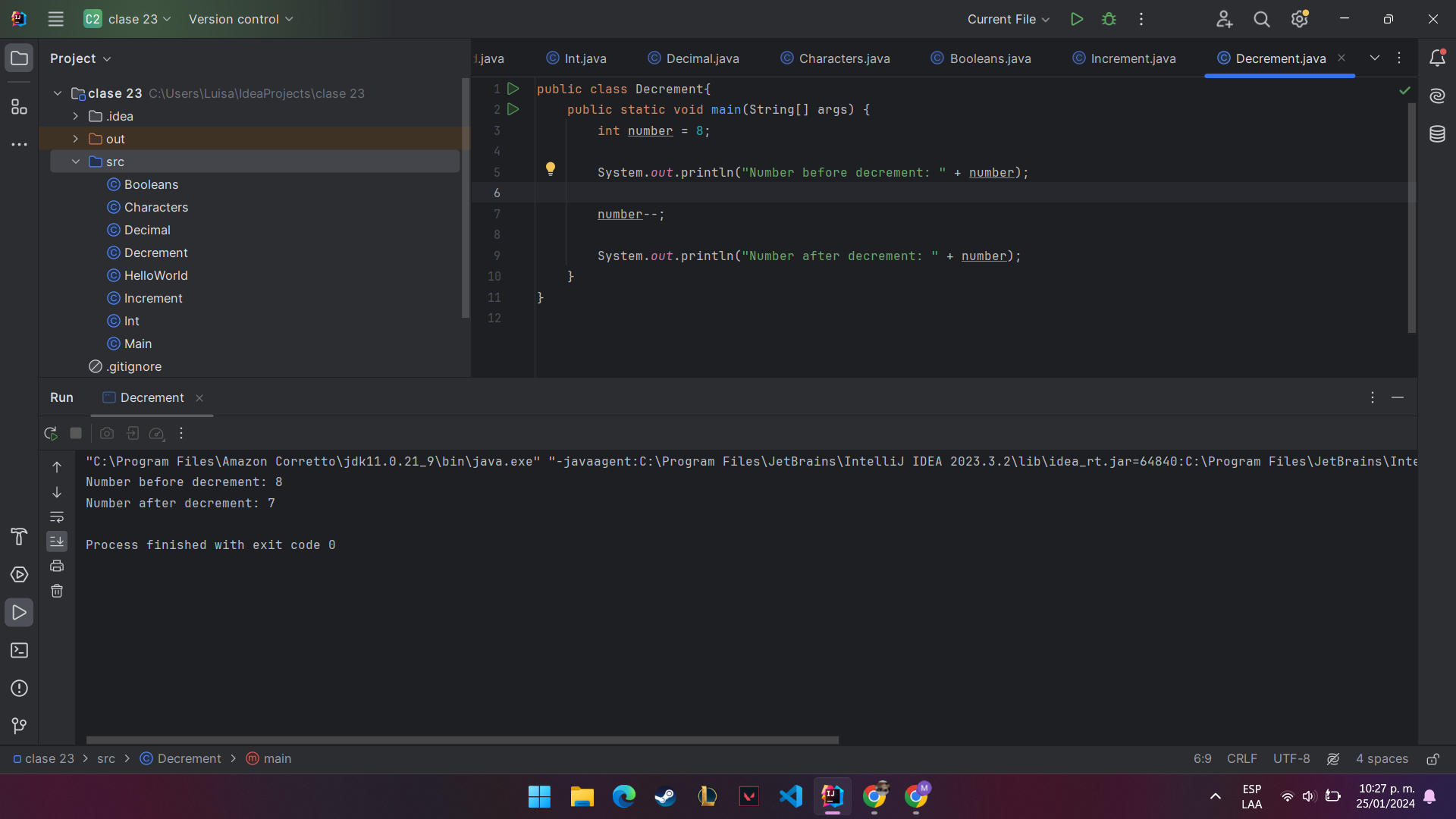
Operador de Incremento (++):

El operador de incremento suma 1 al valor de una variable.



**Decrement Operator (--):**

The decrement operator subtracts 1 from the value of a variable.



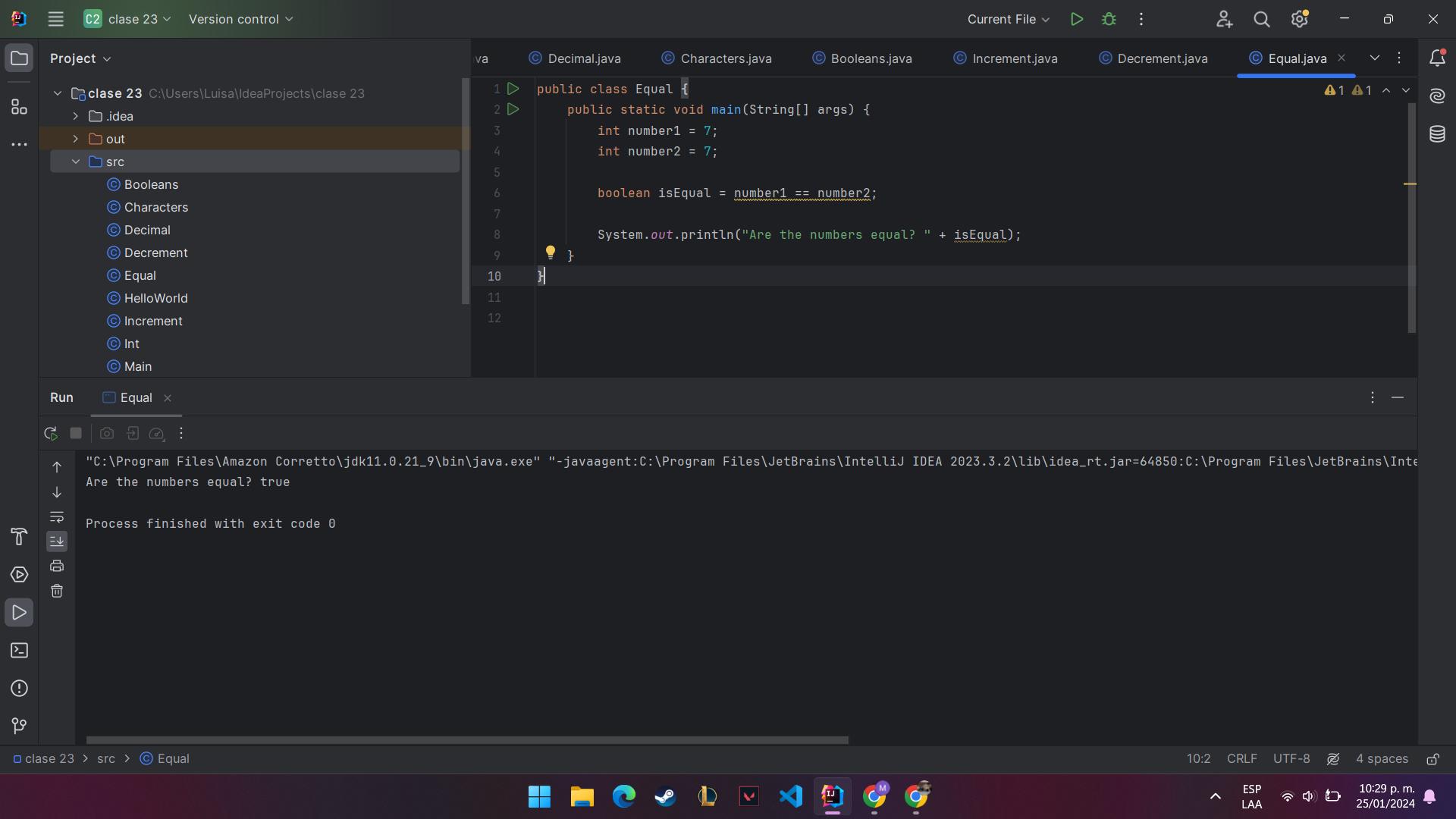
En ambos casos, ya sea de incremento o decremento, puede usar los operadores como post-incremento/decremento (el valor se usa primero y luego se actualiza) o pre-incremento/decremento (el valor se actualiza primero y luego se usa). La elección entre ellos depende de si desea utilizar el valor original o actualizado en la expresión.

Ejercicio #6.

Los operadores de comparación en Java le permiten comparar dos valores y determinar si son iguales, no iguales, mayores, menores, mayores o iguales, o menores o iguales.

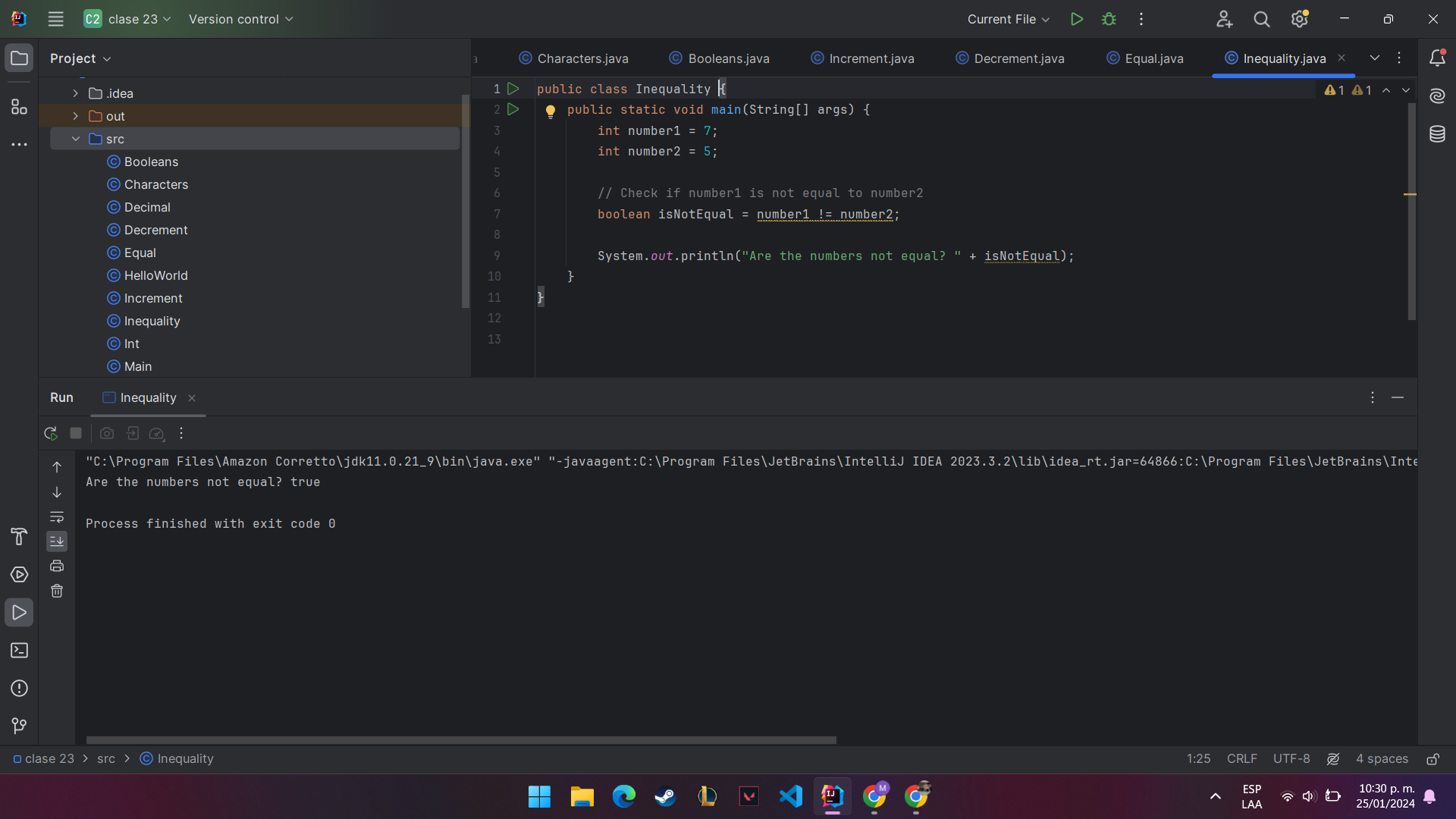
**1. Equality Operator (==):**

El operador de igualdad comprueba si dos valores son iguales.

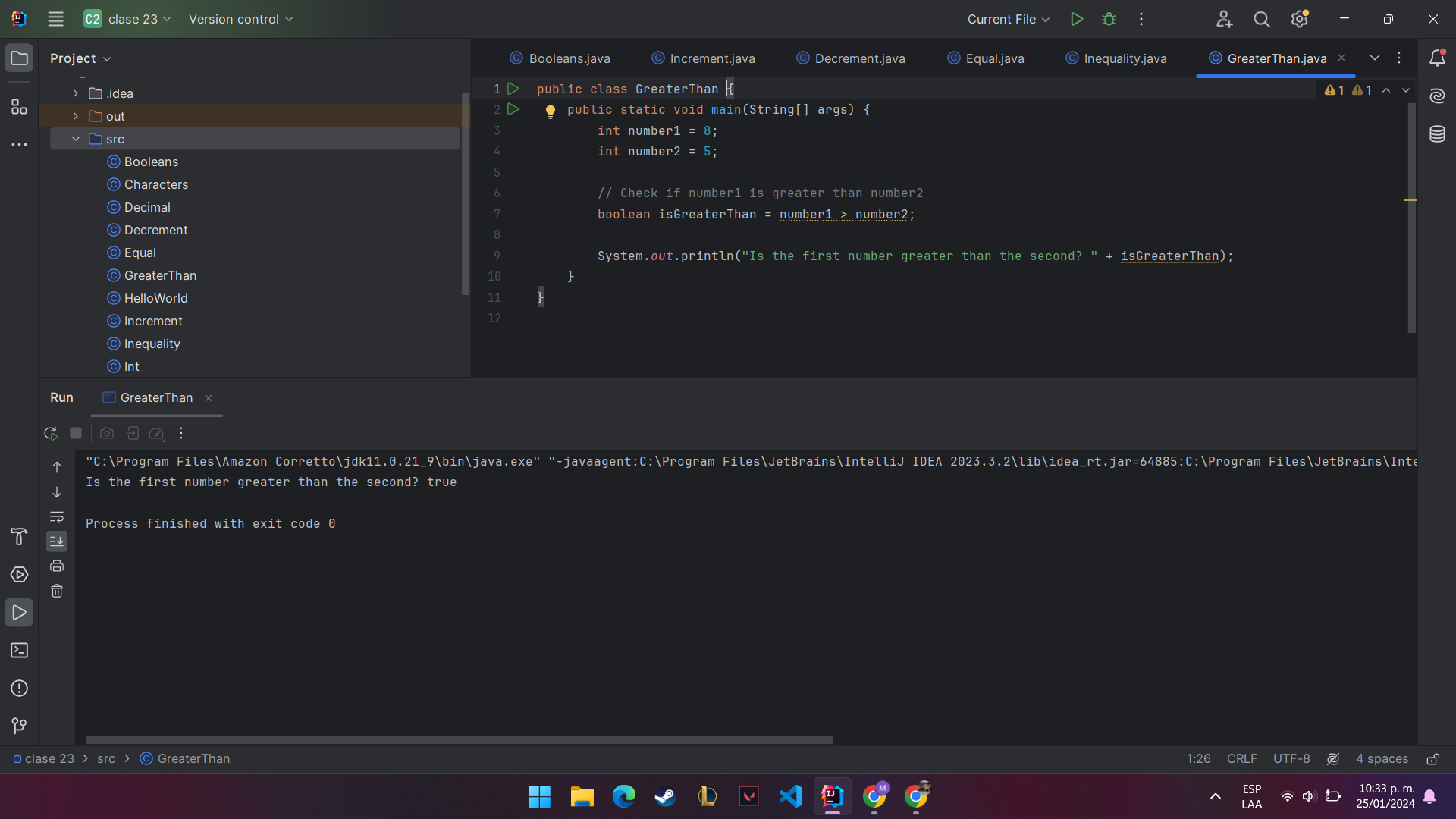


**2. Inequality Operator (!=):**

The inequality operator checks if two values are not equal.

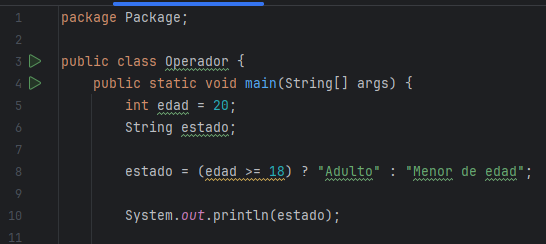


**3. Greater Than Operator (>):**

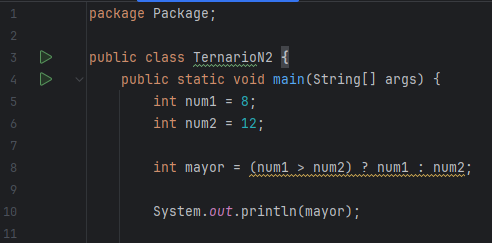
El operador mayor que comprueba si un valor es mayor que otro.

Ejercicio #7.

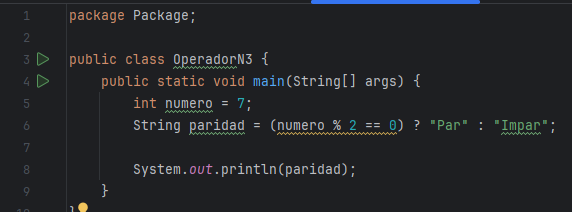
**EJEMPLOS DE USO DE OPERADORES TERNARIOS**

#1  


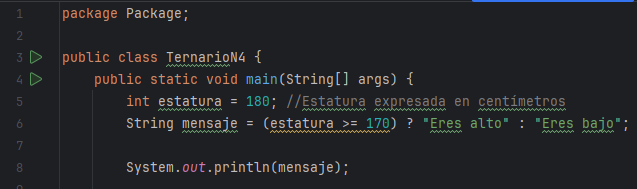
#2.



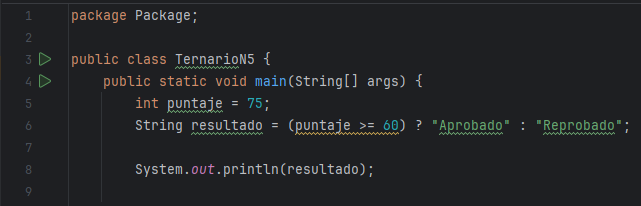
#3.



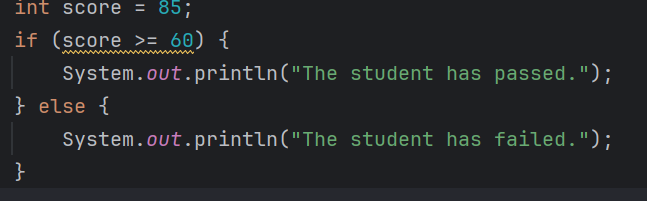
#4.



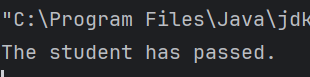
#5.



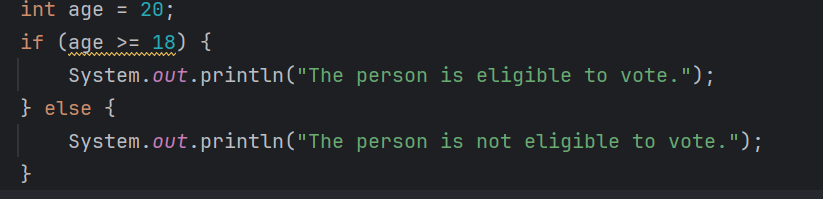
Ejercicio #8.

a.**Ver si un estudiante aprueba o no.**

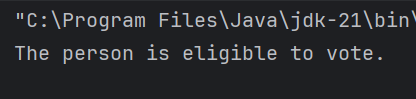
**Salida:**



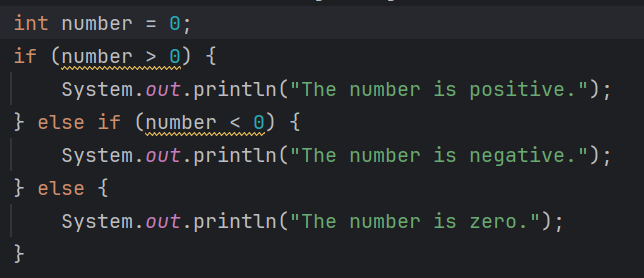
b.**Determinar si una persona puede votar.**



**Salida:**



c.**Ver si un número es positivo, negativo o si es un cero.**

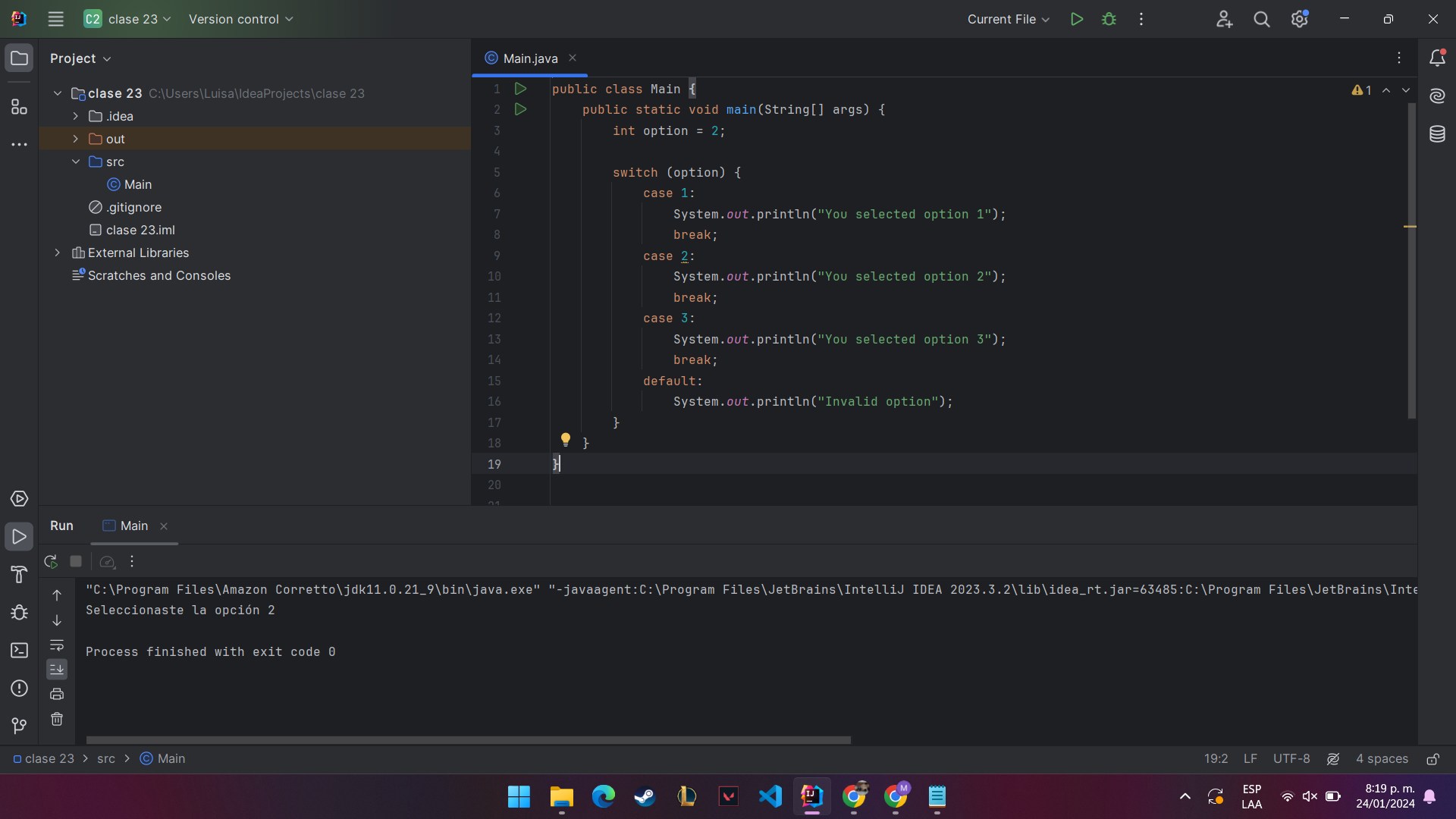
****

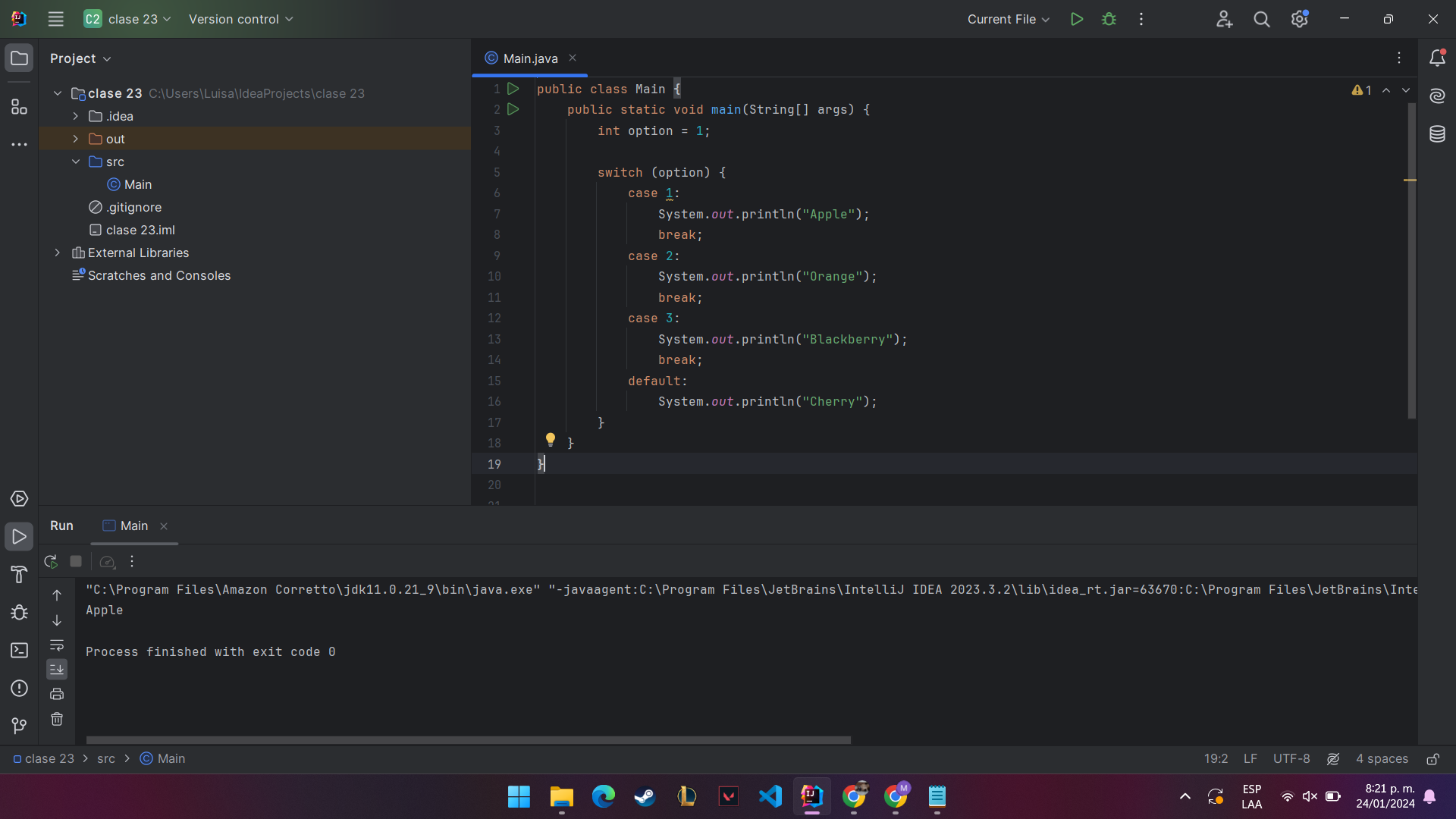
**Salida:**



Ejercicio #9.

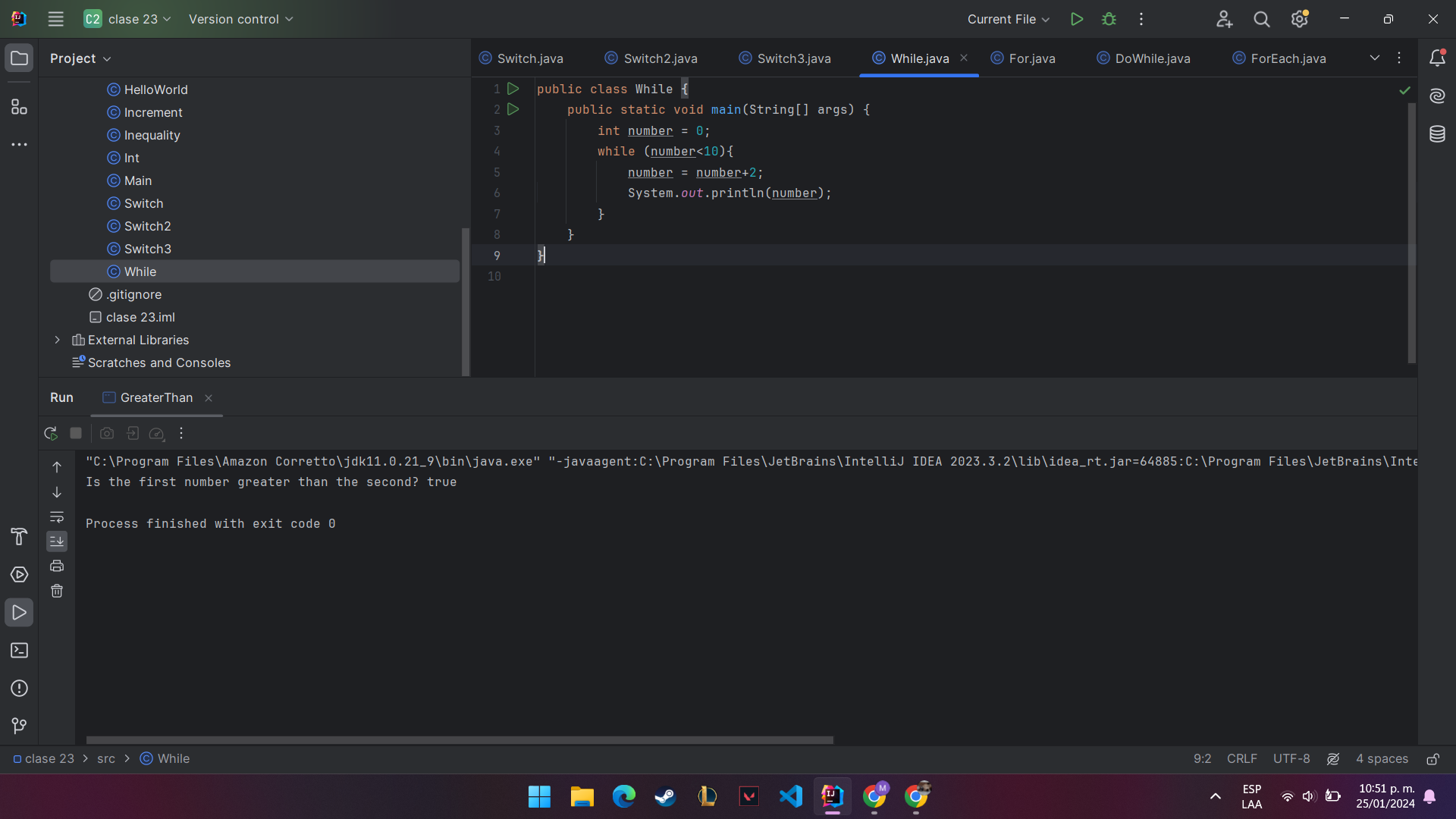
La sentencia switch en Java se utiliza para realizar selecciones múltiples basadas en el valor de una expresión. Proporciona una manera más limpia y eficiente de manejar múltiples casos que usar múltiples sentencias if-else.

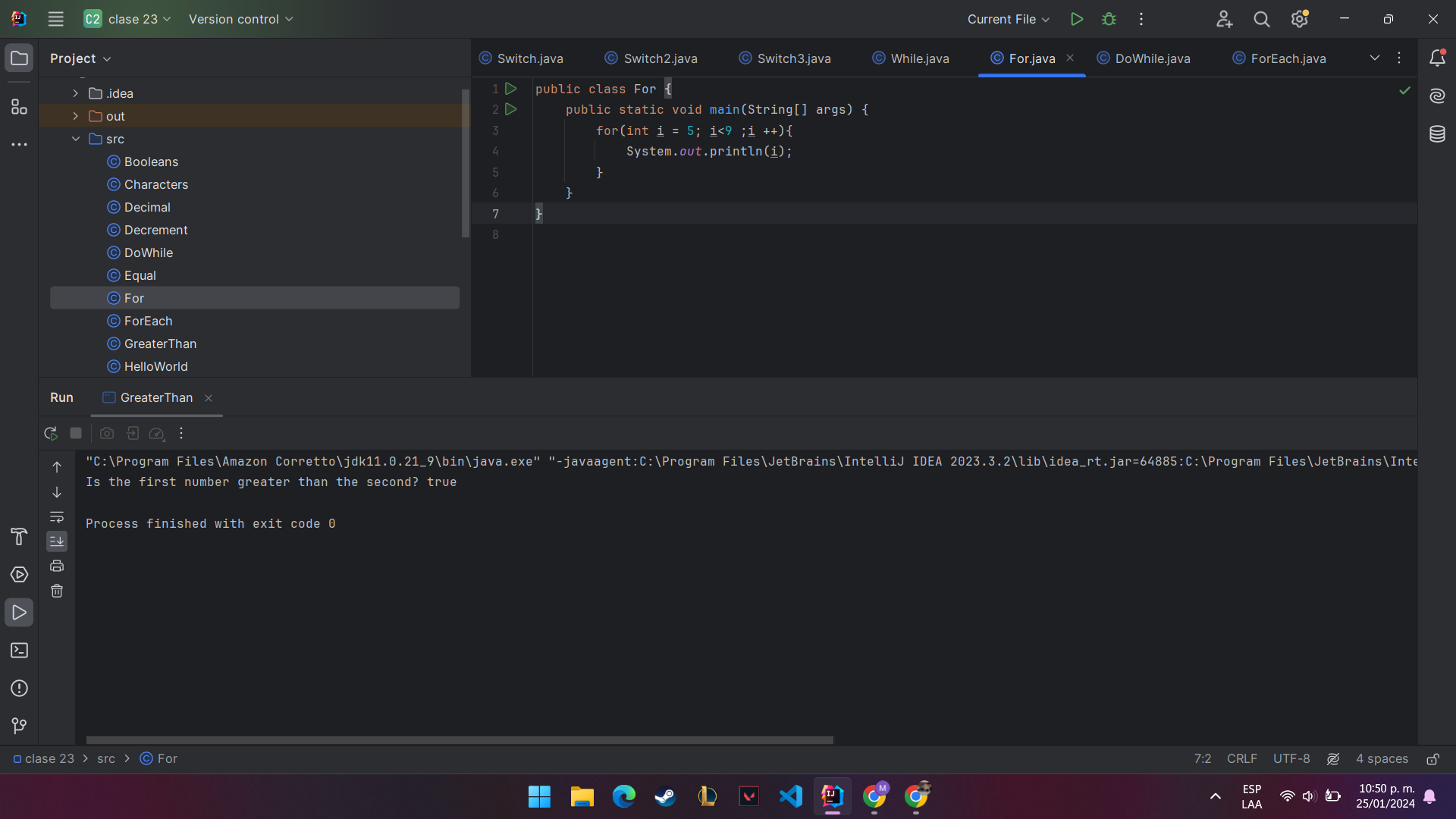


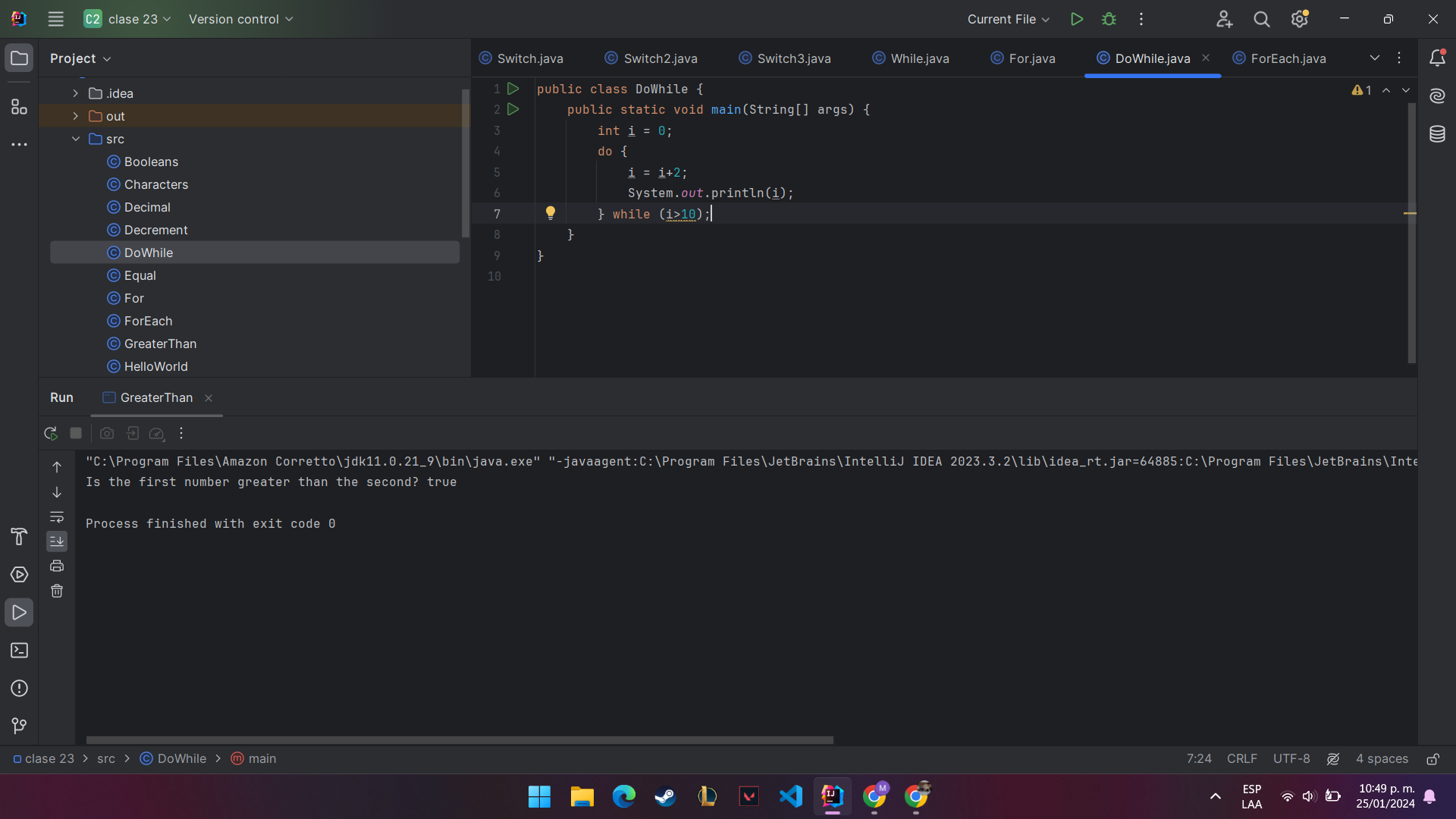




Ejercicio #10.

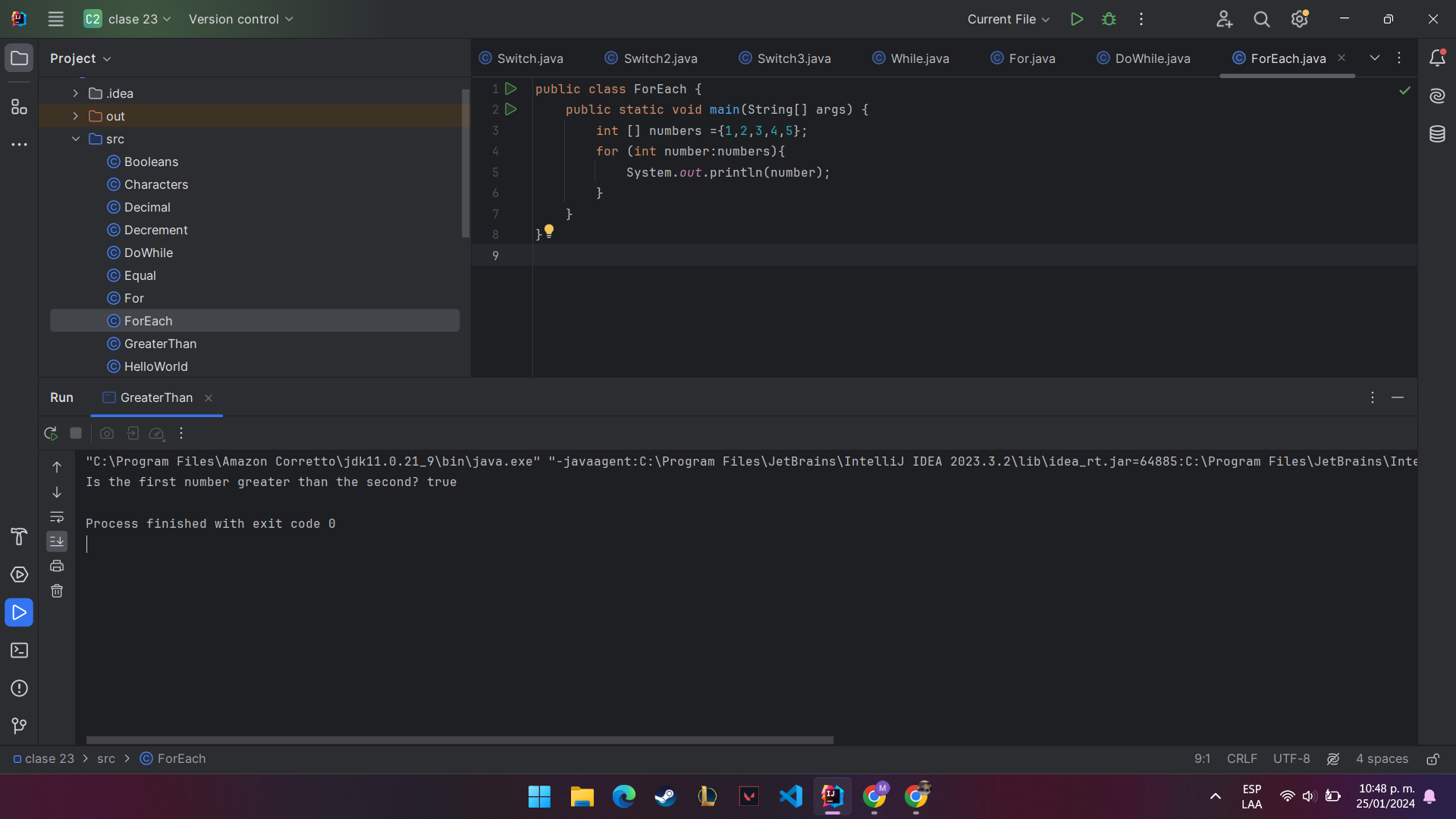






Ejercicio #11.

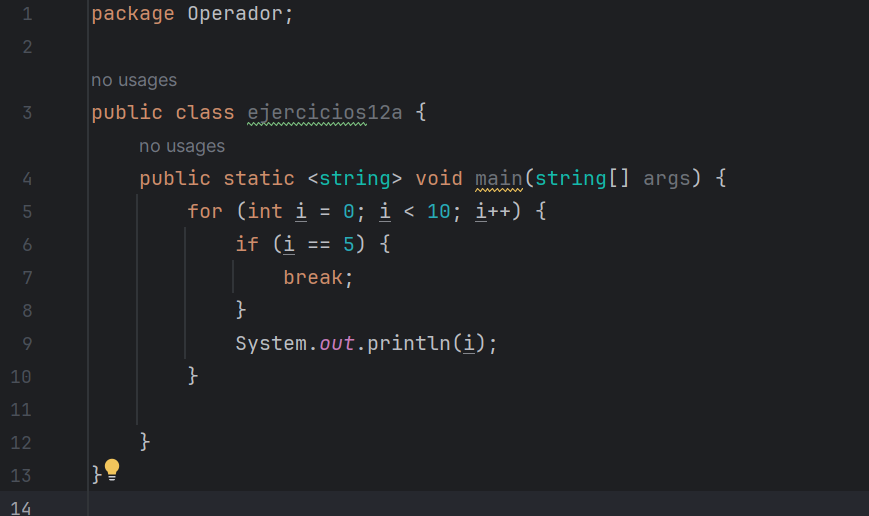
Ejemplo de The foreach Statement.



Ejercicio #12.

Las sentencias break y continue en Java se utilizan para controlar el flujo de los bucles (for, while o do-while).

**La sentencia** break se utiliza para terminar el bucle de manera inmediata. Cuando se encuentra una sentencia break en un bucle, el control salta inmediatamente después del bucle, y el bucle se termina.



**La sentencia** continue, por otro lado, salta a la siguiente iteración del bucle. Es decir, se salta las sentencias restantes en el cuerpo del bucle y continúa con la siguiente iteración.



Ejercicio #13.

Captura de pantalla de depuración de Java

