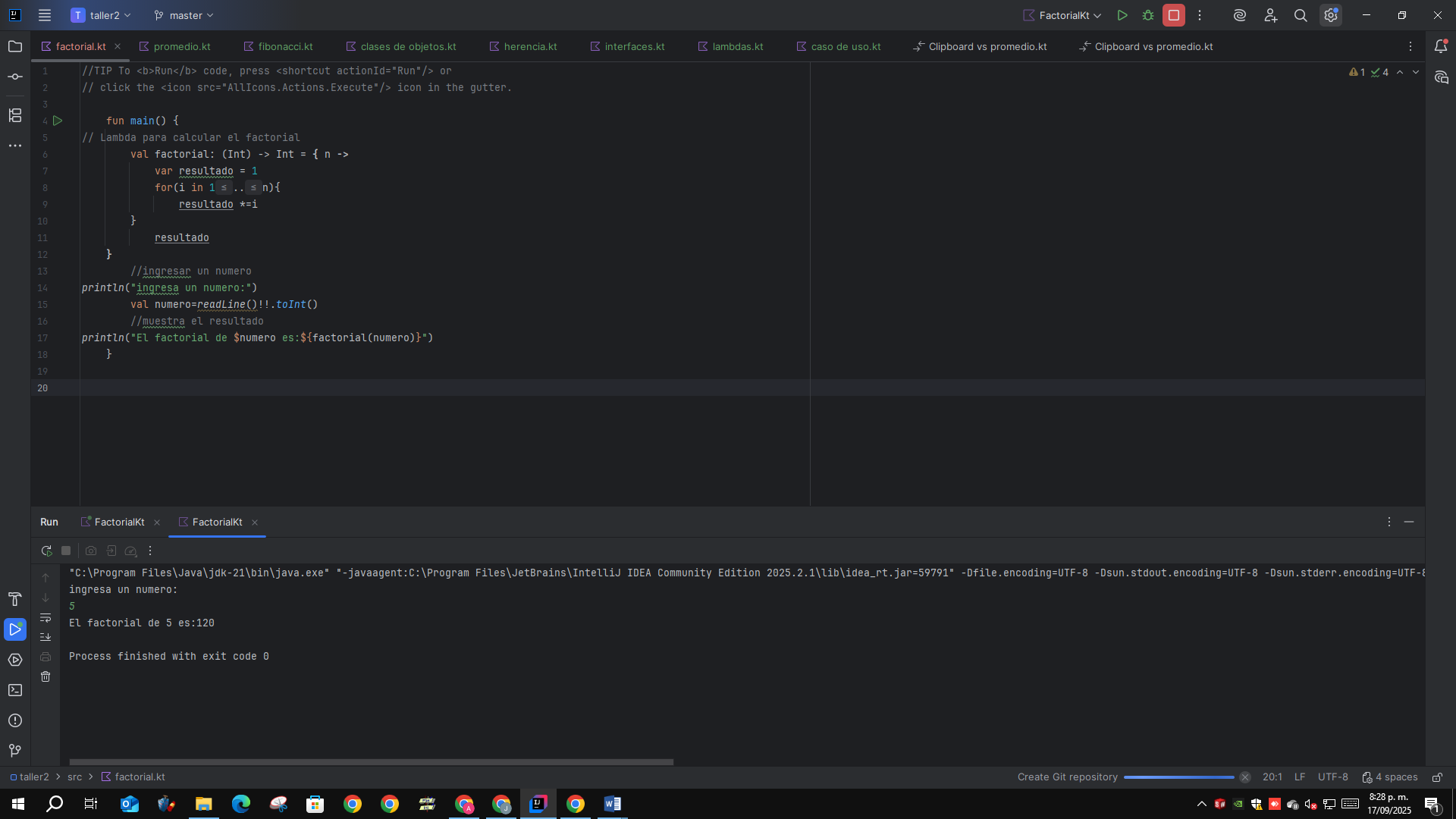
Respuestas Taller 2 - Kotlin

# Ejercicio 1: Factorial de un Número

Este ejercicio solicita calcular el factorial de un número ingresado por el usuario.  
El programa utiliza un ciclo for para multiplicar los valores desde 1 hasta el número dado, acumulando el resultado en una variable factorial.  
Con esto se refuerzan los conceptos de **entrada de datos**, **estructuras repetitivas** y **uso de variables de tipo Long** para manejar números grandes.

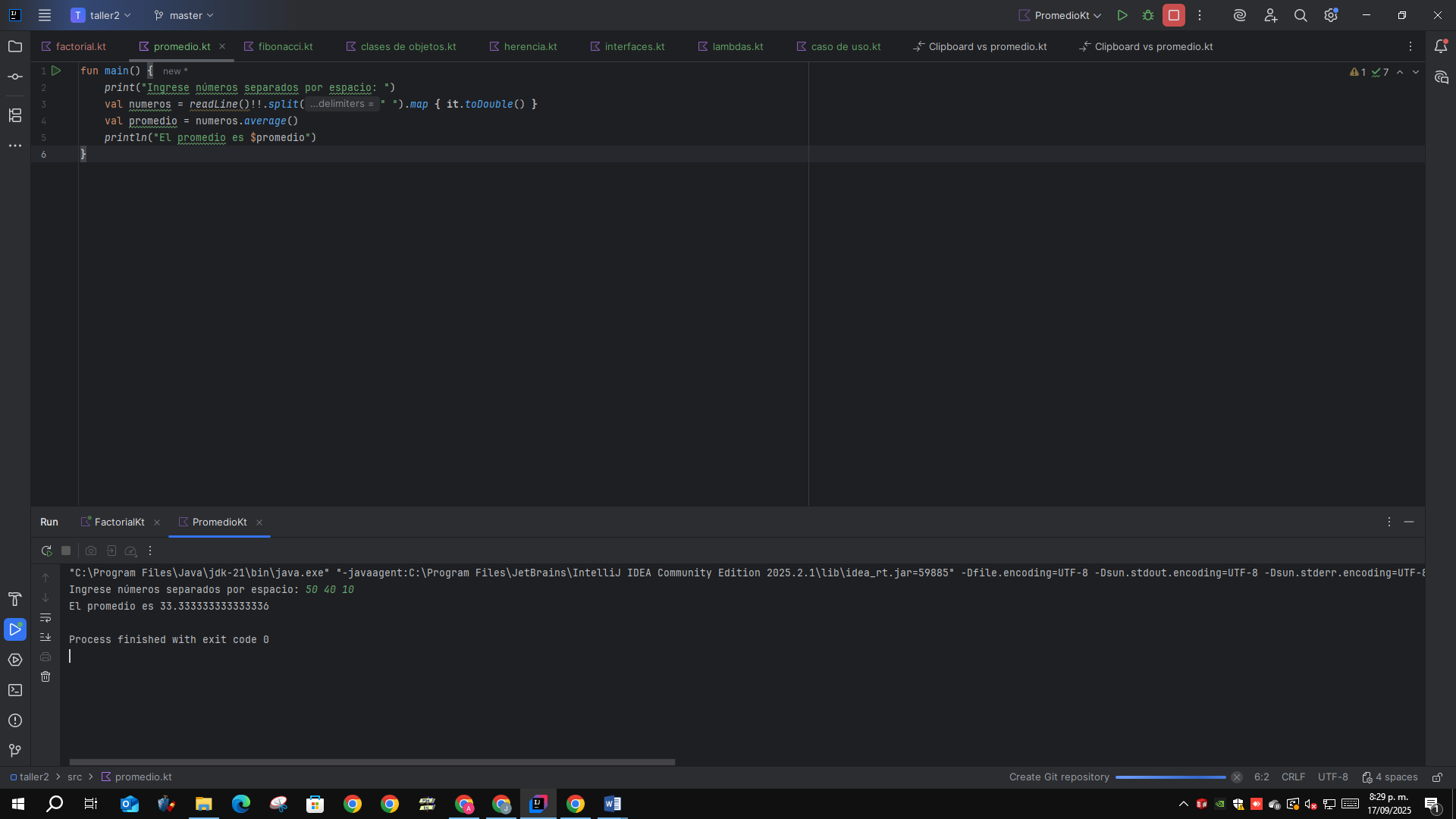
fun main() {  
// Lambda para calcular el factorial  
 val factorial: (Int) -> Int = **{** n **->** var resultado = 1  
 for(i in 1..n){  
 resultado \*=i  
 }  
 resultado  
 **}** //ingresar un numero  
*println*("ingresa un numero:")  
 val numero=*readLine*()!!.*toInt*()  
 //muestra el resultado  
*println*("El factorial de $numero es:${factorial(numero)}")  
 }



# Ejercicio 2: Promedio de una Lista de Números

El objetivo es calcular el promedio de una lista de números ingresados por el usuario.  
Se usa el método split(" ") para separar los números por espacios, luego map convierte cada entrada en un valor Double. Finalmente, se aplica el método average() para obtener el resultado.  
Este ejercicio permite practicar el **manejo de colecciones en Kotlin** y el **uso de funciones predefinidas**.

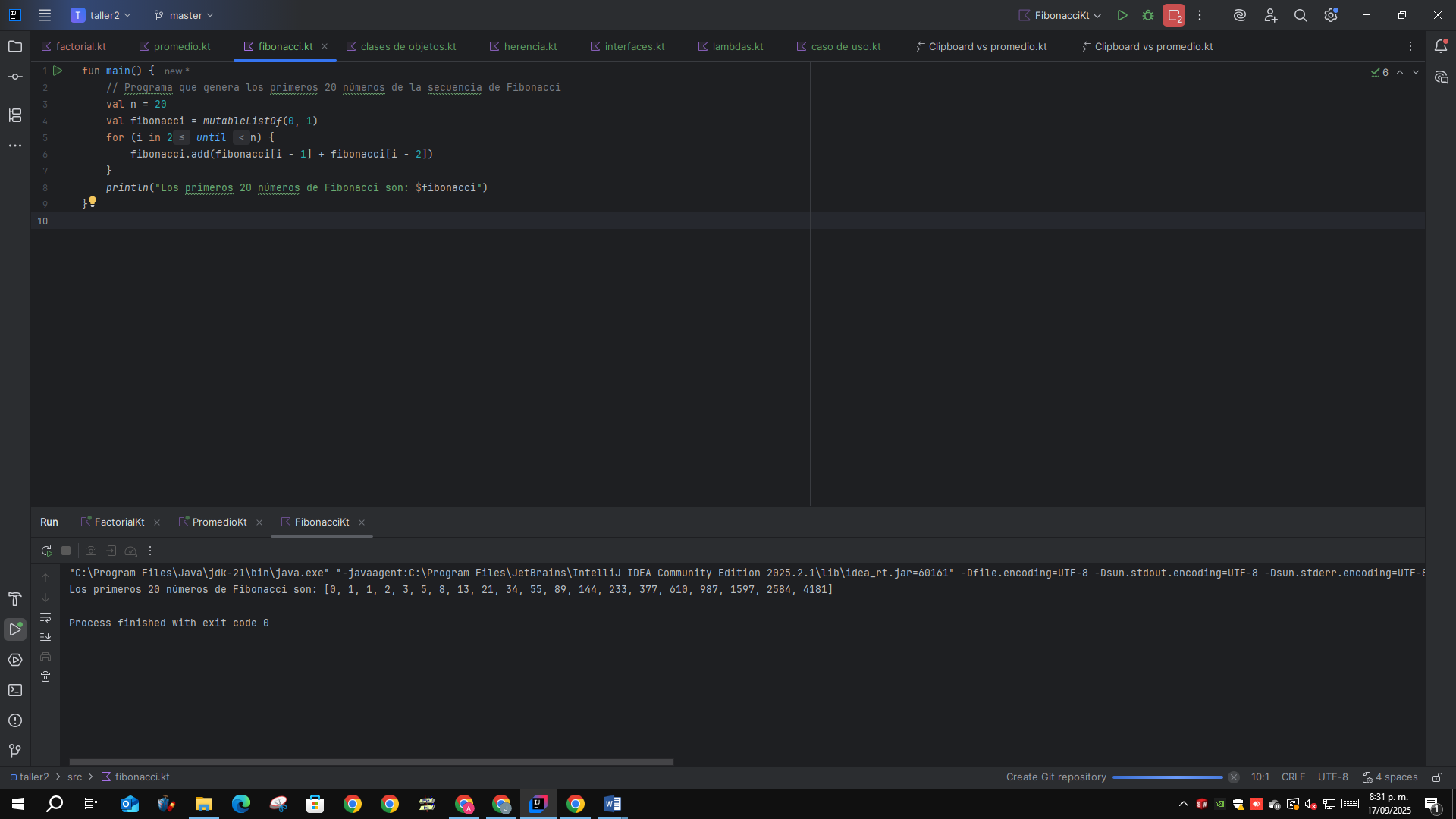
fun main() {  
 *print*("Ingrese números separados por espacio: ")  
 val numeros = *readLine*()!!.*split*(" ").*map* **{ it**.*toDouble*() **}** val promedio = numeros.*average*()  
 *println*("El promedio es $promedio")  
}



# Ejercicio 3: Fibonacci

Se pide generar los primeros 20 números de la secuencia de Fibonacci.  
La solución se basa en una lista mutable (mutableListOf) donde se van agregando los valores sumando los dos anteriores.  
Es un buen ejemplo para reforzar **estructuras de listas dinámicas** y la **lógica de generación de secuencias**.

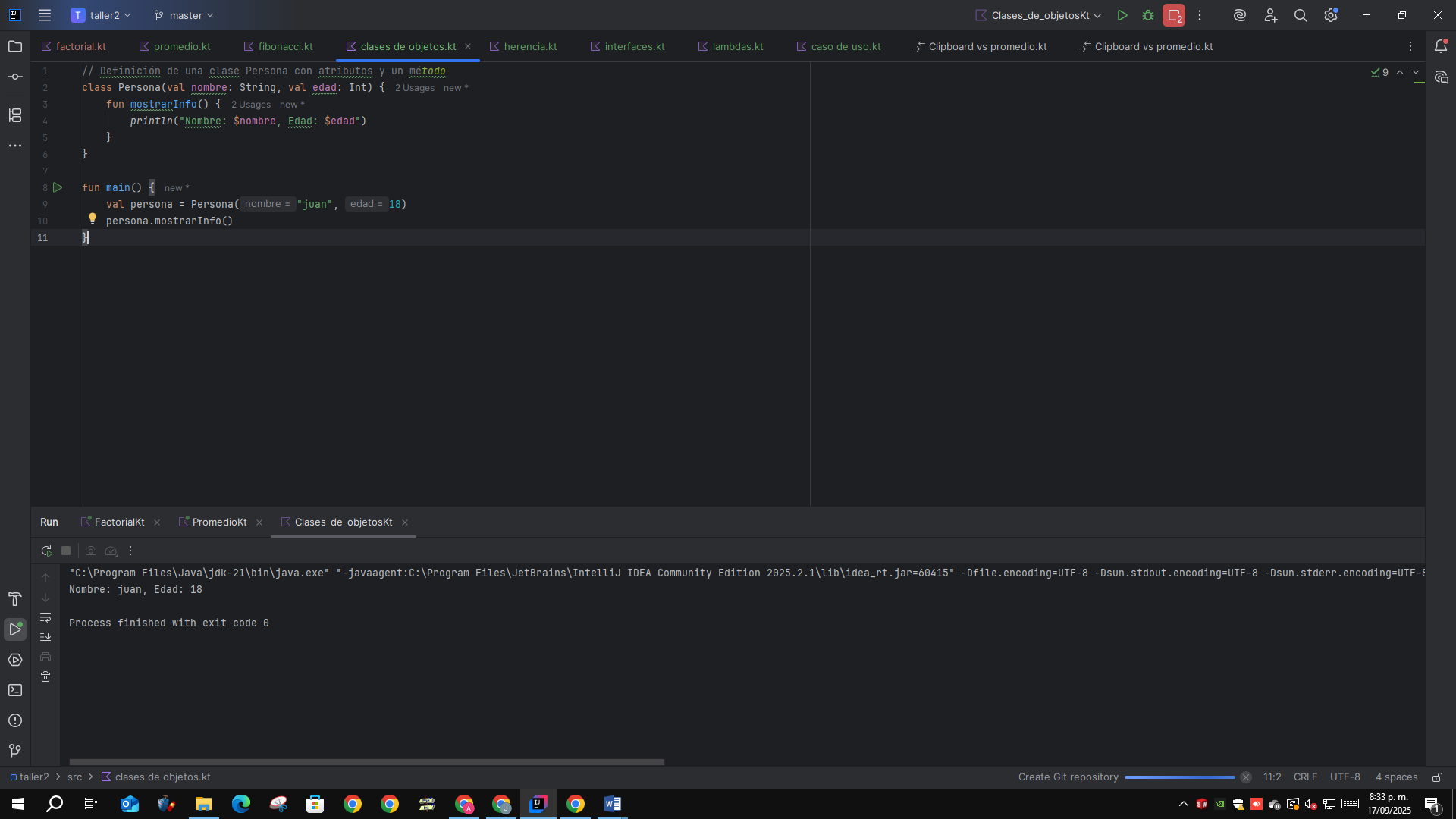
fun main() {  
 // Programa que genera los primeros 20 números de la secuencia de Fibonacci  
 val n = 20  
 val fibonacci = *mutableListOf*(0, 1)  
 for (i in 2 *until* n) {  
 fibonacci.add(fibonacci[i - 1] + fibonacci[i - 2])  
 }  
 *println*("Los primeros 20 números de Fibonacci son: $fibonacci")  
}



# Ejercicio 4: Clases y Objetos

Aquí se define una clase Persona con atributos básicos como nombre y edad, y un método mostrarInfo() que imprime su información.  
El ejercicio introduce los conceptos de **programación orientada a objetos (POO)** en Kotlin: creación de clases, instanciación de objetos y uso de métodos.

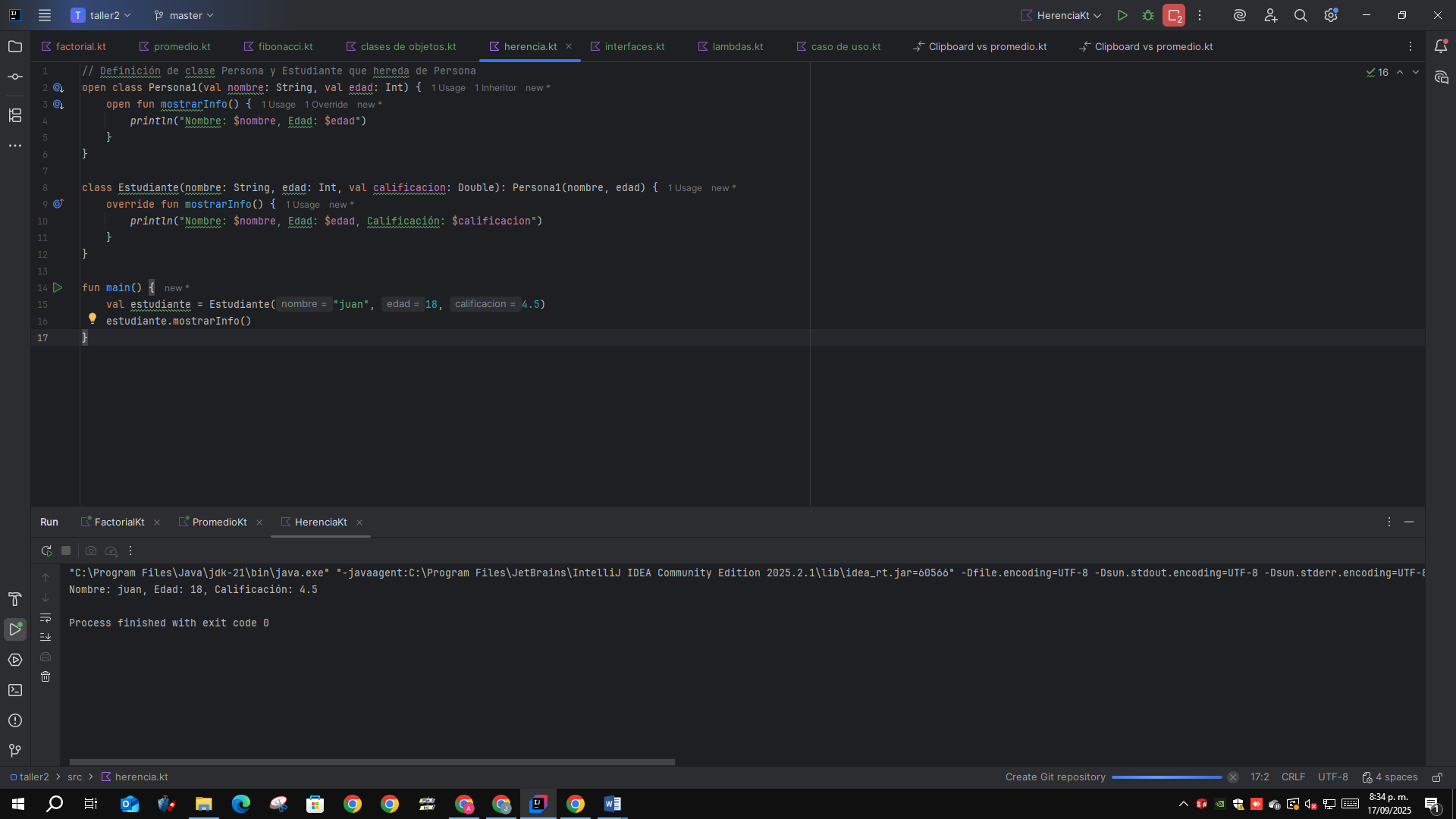
class Persona(val nombre: String, val edad: Int) {  
 fun mostrarInfo() {  
 *println*("Nombre: $nombre, Edad: $edad")  
 }  
}  
  
fun main() {  
 val persona = Persona("juan", 18)  
 persona.mostrarInfo()  
}



# Ejercicio 5: Herencia

Se amplía el ejercicio anterior, ahora creando una clase Estudiante que hereda de Persona e incluye un atributo adicional de calificación.  
Se hace uso de open para permitir herencia y de override para redefinir el método mostrarInfo().  
Este ejercicio explica la **herencia en POO** y el **polimorfismo** en Kotlin.

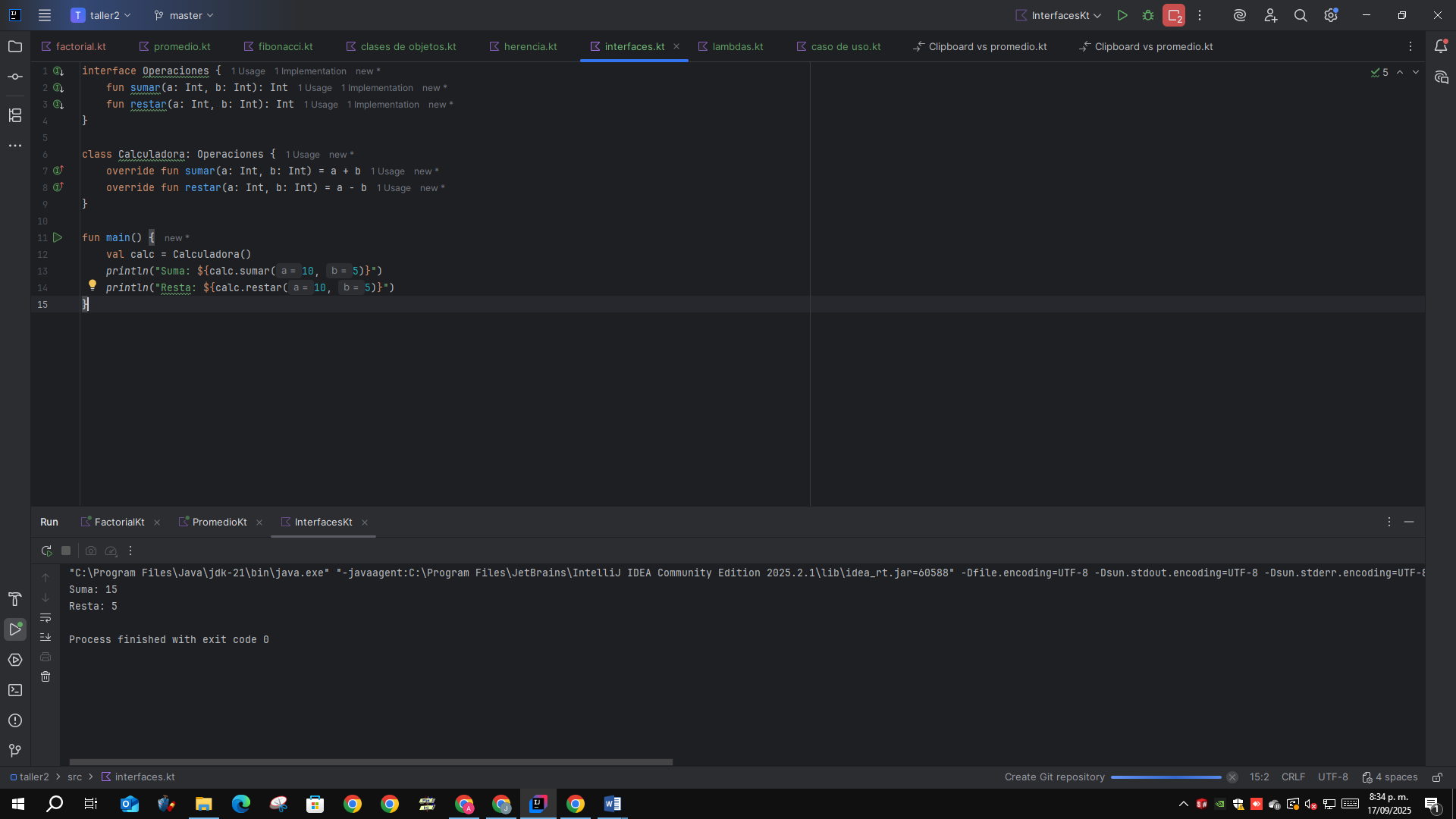
open class Persona1(val nombre: String, val edad: Int) {  
 open fun mostrarInfo() {  
 *println*("Nombre: $nombre, Edad: $edad")  
 }  
}  
  
class Estudiante(nombre: String, edad: Int, val calificacion: Double): Persona1(nombre, edad) {  
 override fun mostrarInfo() {  
 *println*("Nombre: $nombre, Edad: $edad, Calificación: $calificacion")  
 }  
}  
  
fun main() {  
 val estudiante = Estudiante("juan", 18, 4.5)  
 estudiante.mostrarInfo()  
}



# Ejercicio 6: Interfaces

Se define una interfaz Operaciones con métodos para sumar y restar, y una clase Calculadora que implementa la interfaz.  
Esto permite entender cómo en Kotlin se puede lograr la **abstracción** y garantizar que una clase cumpla con un **contrato definido por la interfaz**.  
Ejercicio clave para comprender el **acoplamiento débil y la reutilización de código**.

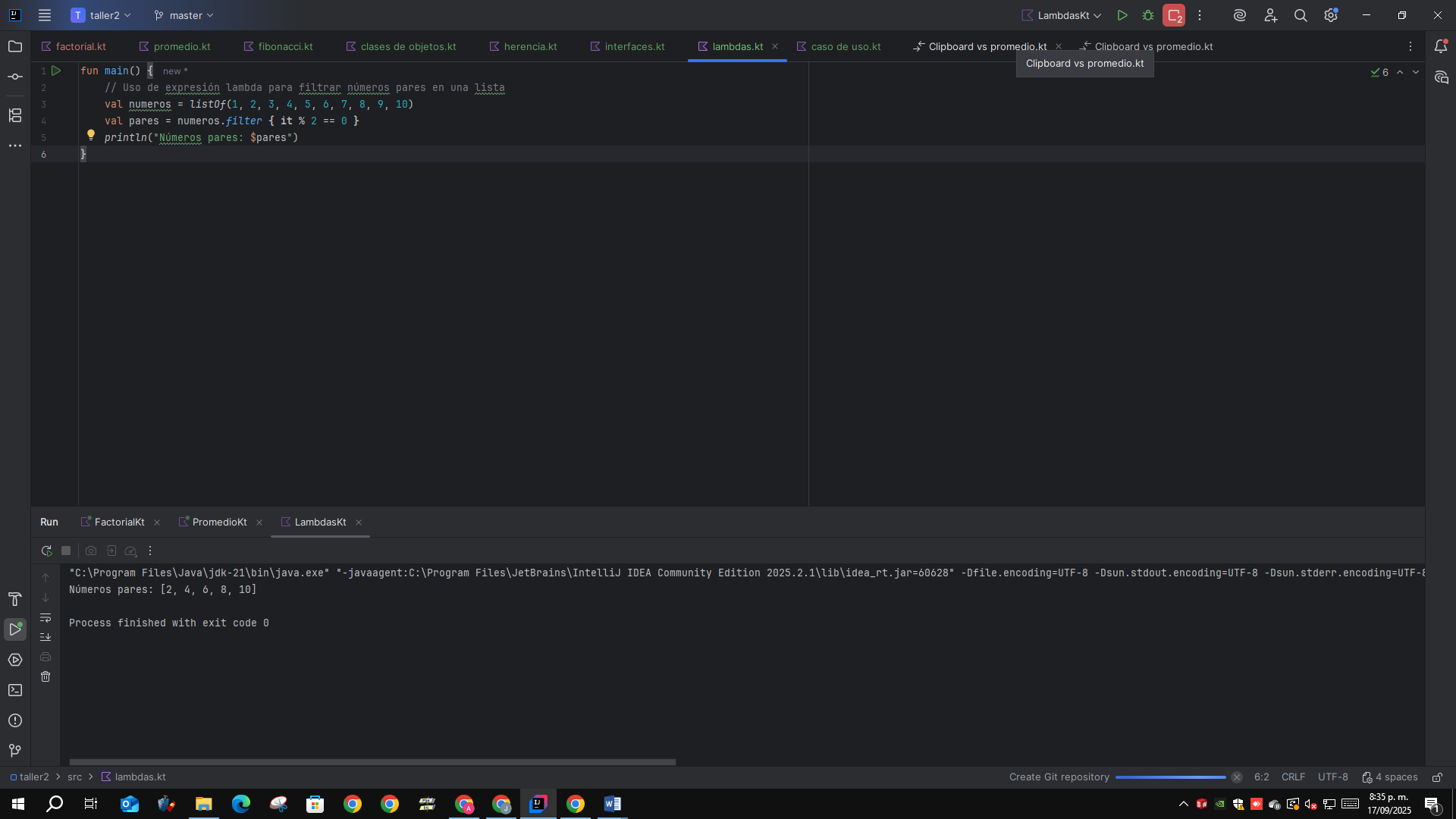
interface Operaciones {  
 fun sumar(a: Int, b: Int): Int  
 fun restar(a: Int, b: Int): Int  
}  
  
class Calculadora: Operaciones {  
 override fun sumar(a: Int, b: Int) = a + b  
 override fun restar(a: Int, b: Int) = a - b  
}  
  
fun main() {  
 val calc = Calculadora()  
 *println*("Suma: ${calc.sumar(10, 5)}")  
 *println*("Resta: ${calc.restar(10, 5)}")  
}



# Ejercicio 7: Lambdas

Este ejercicio utiliza una **expresión lambda** para filtrar los números pares de una lista.  
Se emplea el método filter con la condición it % 2 == 0.  
El objetivo es mostrar cómo las **funciones de orden superior** y las **lambdas** hacen el código más conciso y legible.

fun main() {  
 // Uso de expresión lambda para filtrar números pares en una lista  
 val numeros = *listOf*(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
 val pares = numeros.*filter* **{ it** % 2 == 0 **}** *println*("Números pares: $pares")  
}



# Ejercicio 8: Caso de Uso - App de Tareas

Se desarrolla un sistema sencillo para gestionar tareas, con las siguientes funciones:

* **Agregar tareas** con una descripción.
* **Visualizar todas las tareas** registradas.
* **Marcar como completadas** las tareas elegidas.

Para ello se usa una data class llamada Tarea, que representa cada actividad, y una clase GestorTareas que maneja la lista.  
Este caso práctico integra varios conceptos vistos: **colecciones, clases, métodos, estados mutables y bucles**, mostrando cómo se pueden aplicar en un escenario real.

// App de gestión de tareas simples en consola  
data class Tarea(val descripcion: String, var completada: Boolean = false)  
  
class GestorTareas {  
 private val tareas = *mutableListOf*<Tarea>()  
  
 fun agregarTarea(descripcion: String) {  
 tareas.add(Tarea(descripcion))  
 }  
  
 fun verTareas() {  
 tareas.*forEachIndexed* **{** index, tarea **->** *println*("${index + 1}. ${tarea.descripcion} - ${if (tarea.completada) "Completada" else "Pendiente"}")  
 **}** }  
  
 fun completarTarea(indice: Int) {  
 if (indice in 1..tareas.size) {  
 tareas[indice - 1].completada = true  
 }  
 }  
}  
  
fun main() {  
 val gestor = GestorTareas()  
 gestor.agregarTarea("Estudiar Kotlin")  
 gestor.agregarTarea("Hacer ejercicio")  
 gestor.verTareas()  
 gestor.completarTarea(1)  
 gestor.verTareas()  
}

