# Programación funcional:

Una expresión lambda carecterizada por el símbolo”->” en la asignación de una lógica , es decir código,a una funcon genérica definida en una interfaz.

Los objetos que reciben esta asignación , pertenecientes a la clase del interfaz, se puden llamr como argumentos de otras funciones.

Por ejemplo el objeto suma se recoge mediante la expresión

NumerosEnteros suma=(a,b)->a+b;

Con la funcionalidad de sumar dos números.

La programación funcional es un nuevo paradigma de programación en el cual las funciones se pueden gestionar como objetos , así por ejemplo se pueden asignar funcionas a variables así poderlas enviar como argumentos en la llamada a un método.

Las expresiones lambda se pueden almacenar en variables si el tipo de variable es una interfaz que tiene un solo método. La expresión lambda debe tener la misma cantidad de parámetros y el mismo tipo de retorno que ese método. **Java tiene muchos de estos tipos de interfaces integrados, como la Consumer interfaz (que se encuentra en el java.utilpaquete) utilizada por las listas.**

Se recogen las siguientes interfaces implemnetadas en al librería Java.util.function

Consumer<T> representa una función gnérica que acepta un dato simple y no retorna resultado.

Function<T,R>Representa una fncion genérica que acepta un dato y retorna un resultado

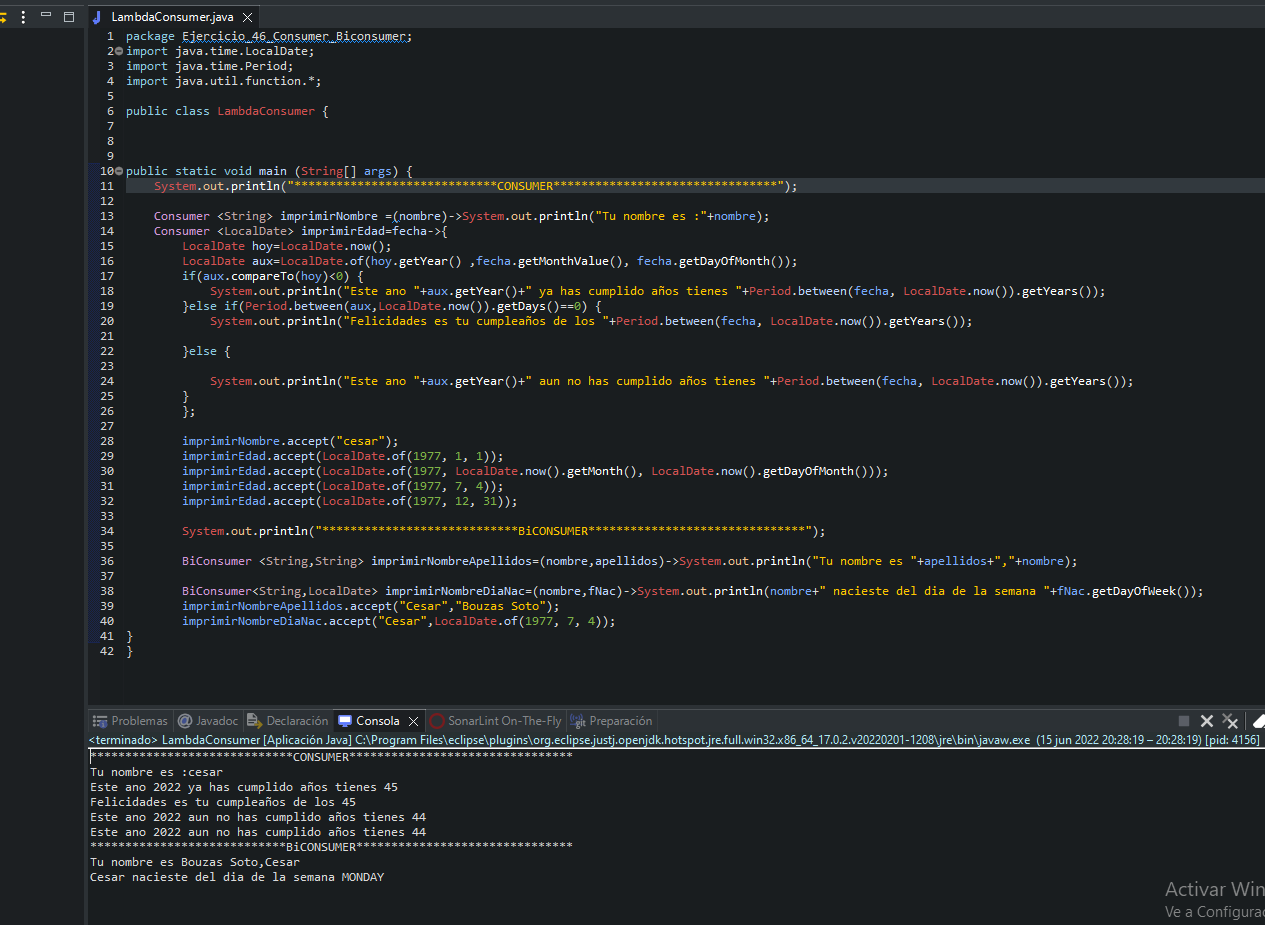
Predicate<T> representa una función bolleana que acepta un dato

Supplier<T> representa un proveedor de resultados.

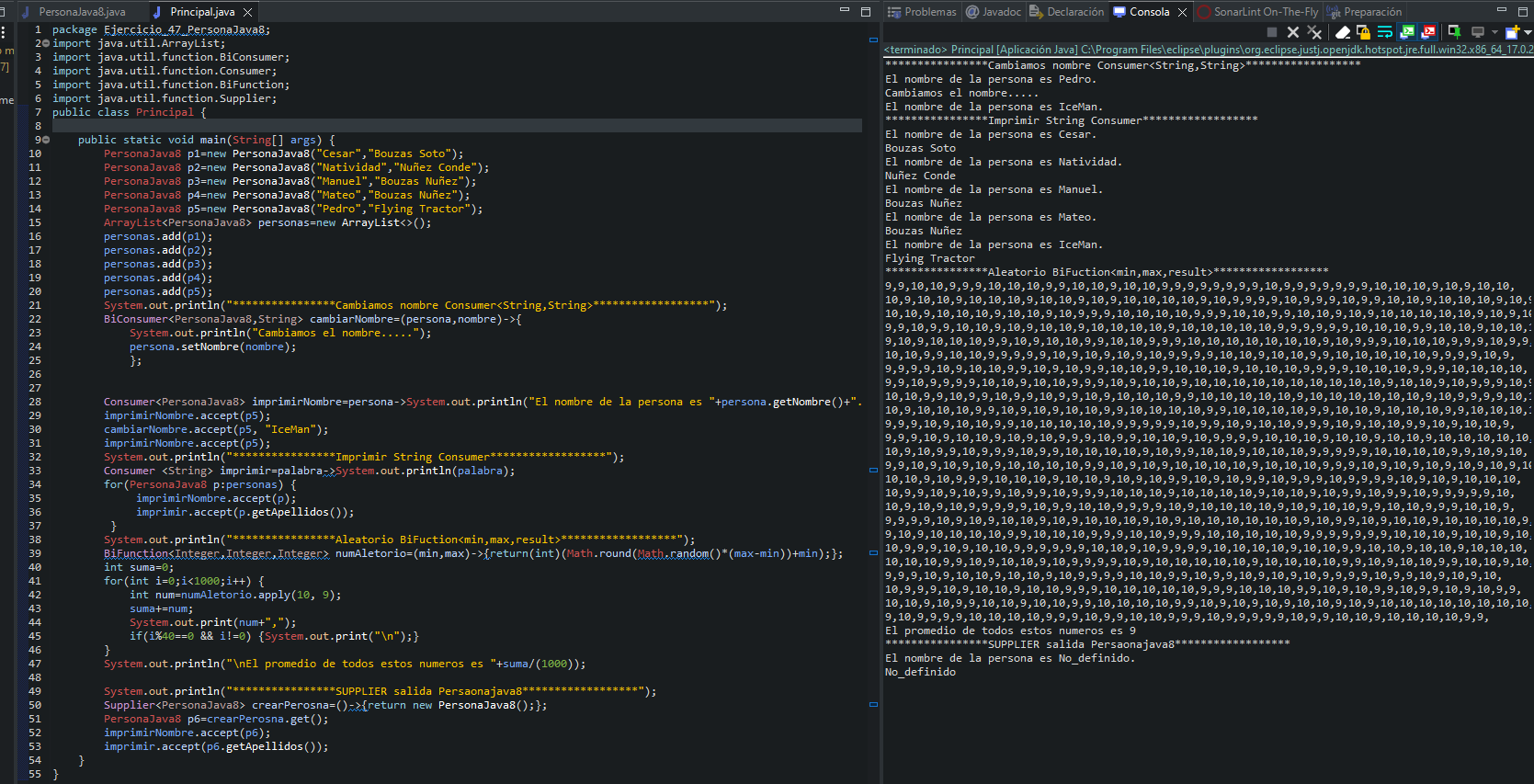
## Haz un ejemplo usando la expresión lambda Consumer y BiConsumer Ejercicio

Las lambdas de tipo **Consumer** reciben por parámetro **un elemento** y no **devuelven nada**.

Las lambdas de tipo **BiConsumer** reciben por parámetro **dos elementos** y no **devuelven nada.**

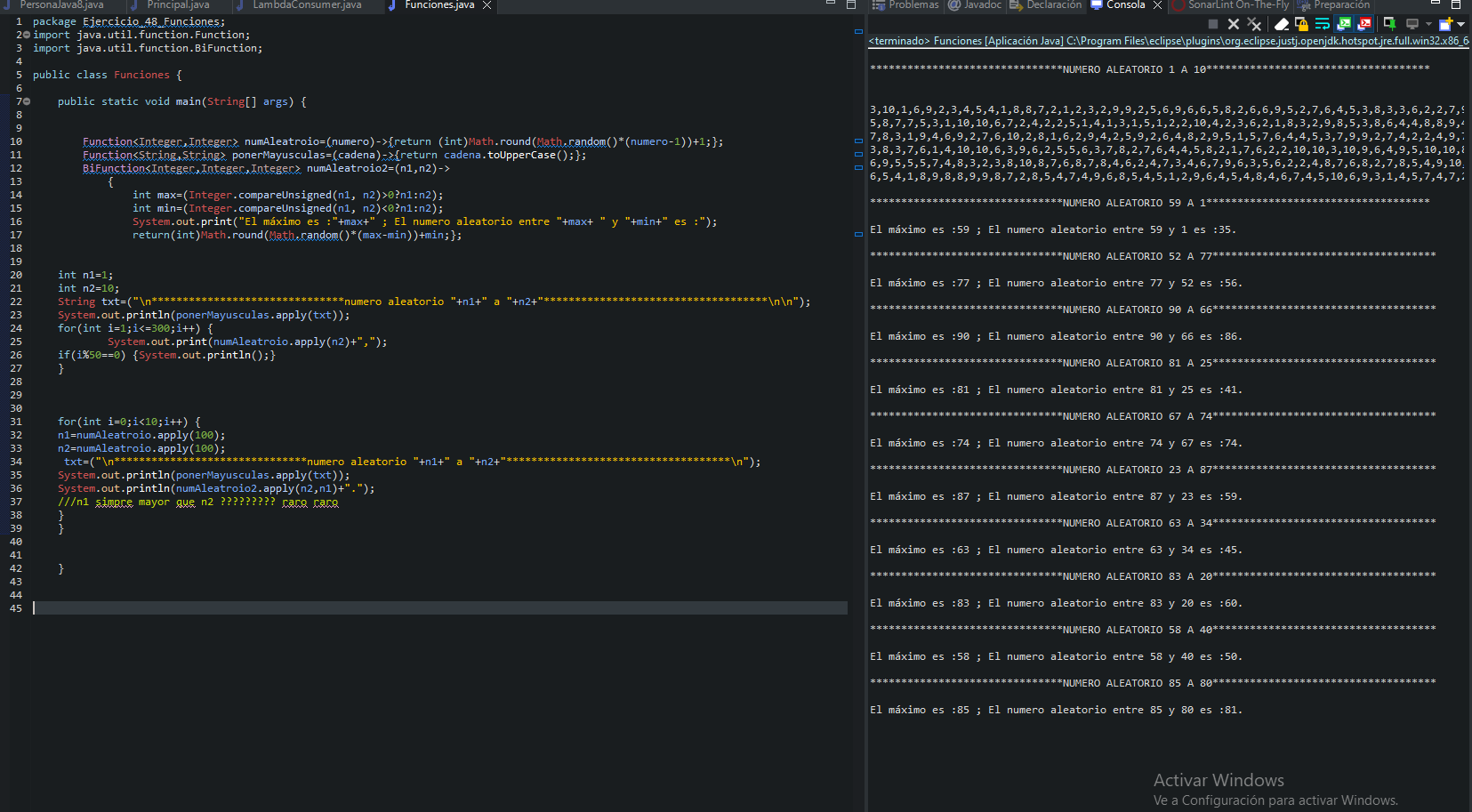


## Crea una clase PersonaJava8 donde tenga un nombre y apellidos. Desde el main crea un método que le cambie el nombre a una persona. Crea un método que te pinte por pantalla un String introducido por parámetro. Llama a este método para mostrar el contenido de un arrayList Haz un método que devuelva un numero al azar. Crea una PersonaJava8 con la expresión Supplier y cambiale el nombre.

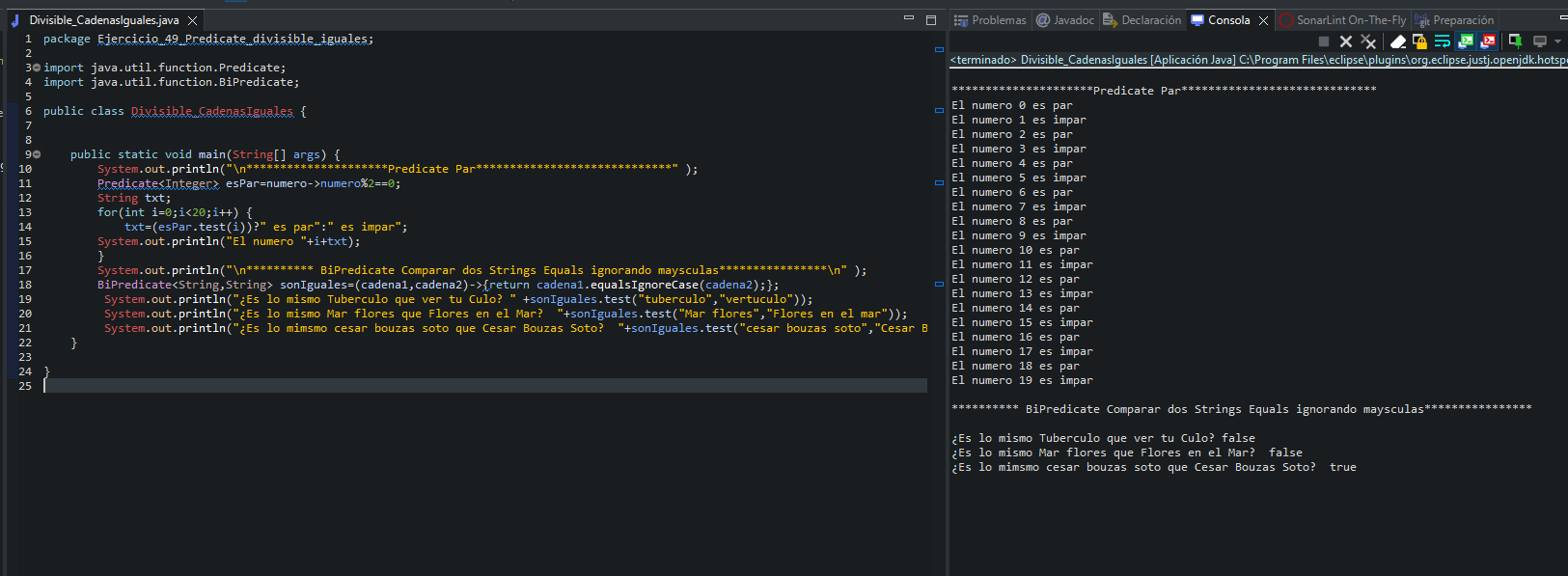


***Trastear con dos puntos…….***

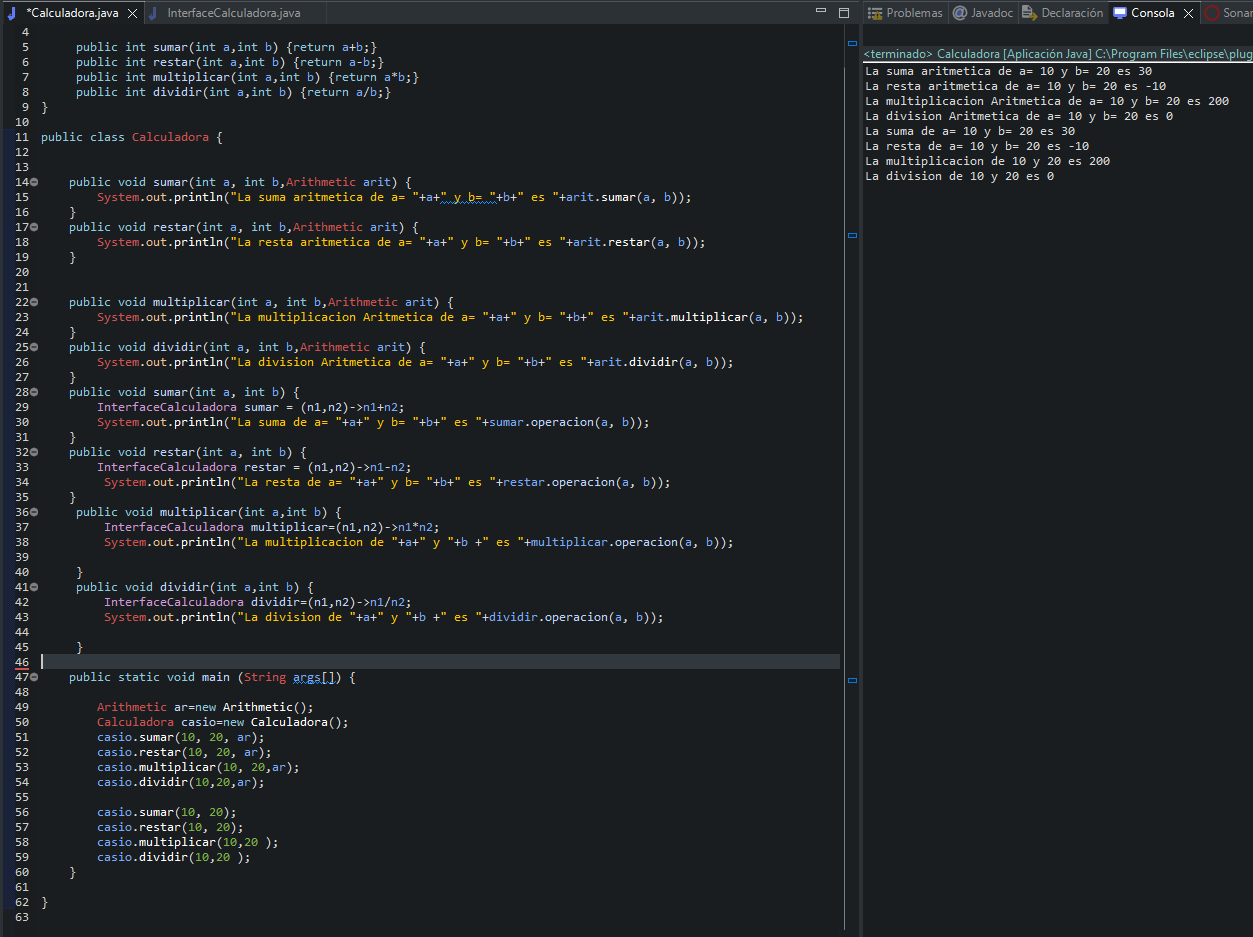
## Crea una expresión Function que devuelva un valor aleatorio entre 1 y un valor introducido por parámetro. Crea una expresión Function que reciba una cadena y la devuelva convertida en mayúsculas. Crea una expresión BiFunction que reciba 2 números, que compare cual es mayor y devuelva un número al azar entre ambos.



## Crea un Predicate que reciba un número y te diga si es divisible entre 2. Crea un BiPredicate que te diga si dos cadenas introducidas son iguales ignorando mayúsculas/minúsculas.



## Crea una interfaz funcional que contenga un método “operacion” al que se le pasen dos números (o doubles). Crea una clase calculadora que tenga un método al que se le pasen dos enteros (o doubles) y un objeto tipo Arithmetic y realice la operación entre ambos valores. Crea otro método que haga lo mismo pero en vez de pasarle un objeto tipo Arithmetic se le pase una BiFuncional. En el main, crea dos operaciones diferentes (suma y resta, por ejemplo) de tipo Arithmetic y creando un objeto tipo calculadora mira las diferentes posibilidades que hay de llamar a sus métodos.



## Crea un Stream de nombres de personas y muestra sus datos por consola (en 2 líneas de código) .

Hay muchas formas de crear un Stream todo stream tiene un origen source, es decir fuente de datos. La fuente puede ser de naturaleza diversa, desde un array , a datos procedentes de un disco o de la tarjeta de red.

Como Stream es una interface, no podemos hacer directamente un new , hay muchas formas de crear un stream , un ejemplo sencillo es el método static of de interface Stream. Desde java 8 las interfaces pueden tener ciertos métodos con código, concretamente son los métodos default y los static. Stream.of() es un método estático de Stream y se usa para crear objetos Stream. Se le pude pasar por parámetro un conjunto de valores separados por comas o un array. En este ejemplo utilizaré la primera de las dos opciones.

## 

## Crea un Stream de nombre de personas en mayúsculas, conviértelo en nombres con solo la primera en mayúscula (el resto en minúsculas), muéstralo por pantalla y luego mételo dentro de una lista.

Otra forma interesante de crear un Stream es crear un Stream con el método stream() que desde java 8 tienen los objetos Collection. Devuelve un Stream secuencial también hay un método para **parallelStream().**

En resumen un método stream que implementa un interface Collection devuelve un objeto Stream que da un flujo de datos ligado a la potencia de la programación funcional.

El método de programación tradicional y funcional son mejor según el contexto pero para el procesado de datos(filtrado ordenado etc.. la programación declarativa(funcional) parece la más conveniente.

El estilo de programación declarativa tipo SQL relacionada con la funcional de java:

Select->map()

From->stream

Where->filter()

Order by->sorted()

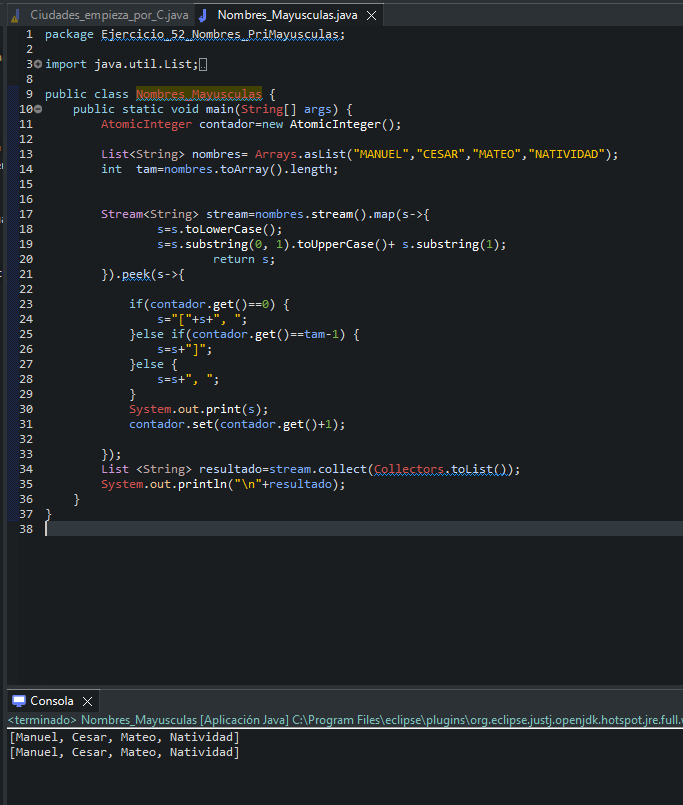
Además observa que en existe una interacción automática sobre cada miembro resultante del from similar a la interacción que se hace en el Stream.

Para este ejemplo necesitamos el mapeo de datos el SELECT , podemos trasformar los elementos de un Stream , para logarlo usamos algunos de los siguientes métodos:

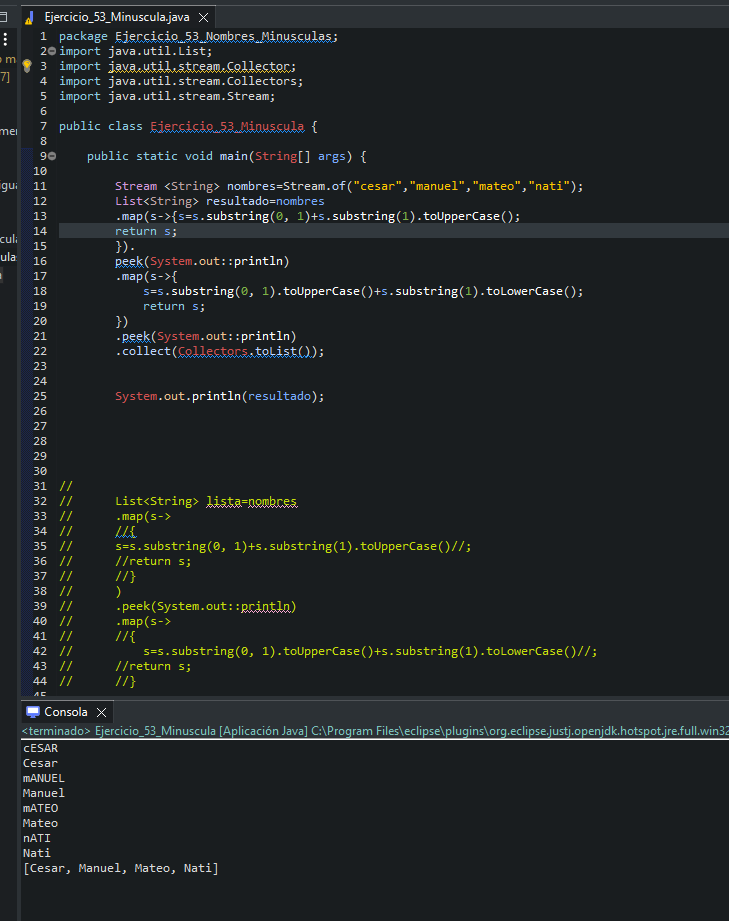
* *map(Function<T,R>):Stream<R>*
* *mapToDouble(TodoubleFunction<T>:DoubleStream*
* *mapToInt(TointFunction<T>:IntStream*
* *mapToLong(ToLongFunction<T>):LongStream*

**Map (Function<T,R>:Stream<R>**

Retorna un Stream que contiene el resultado de aplicar la función pasada por parámetro a todo los elementos del Stream. **Transforma los elementos de Tipo T a tipo R**



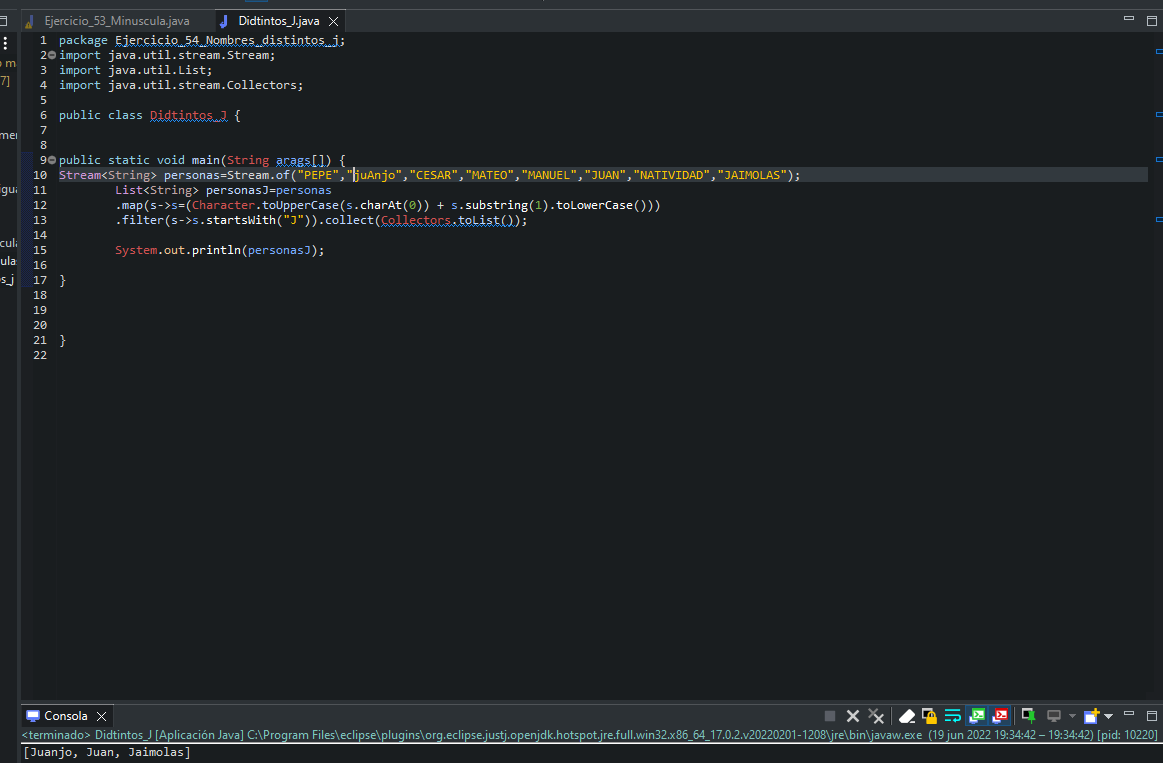
## Crea un Stream de nombre de personas en minúscula, conviértelo en nombres con solo la primera en minúscula , muéstralo por pantalla , luego haz a la inversa (primera mayúscula y las siguientes minúsculas), muéstralo por pantalla y luego mételo dentro de una lista.



No imprime por orden todo junto , Se pude????? Sorted

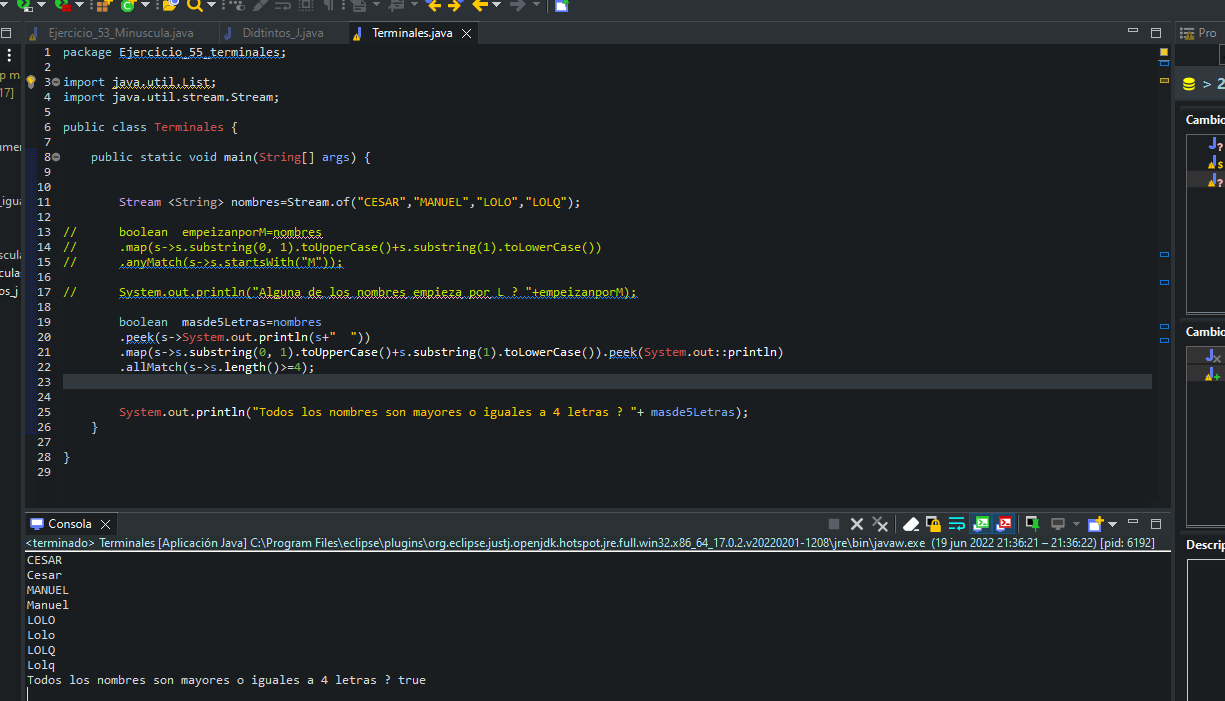
Peek permite ver el estado del Stream en un determinado momento de la aplicación.

## Ejercicio 54: Crea un Stream de nombre de personas en mayúsculas, conviértelo en nombres con solo la primera en mayúscula (el resto en minúsculas), y muestra por pantalla, sólo los nombres que empiecen por J.



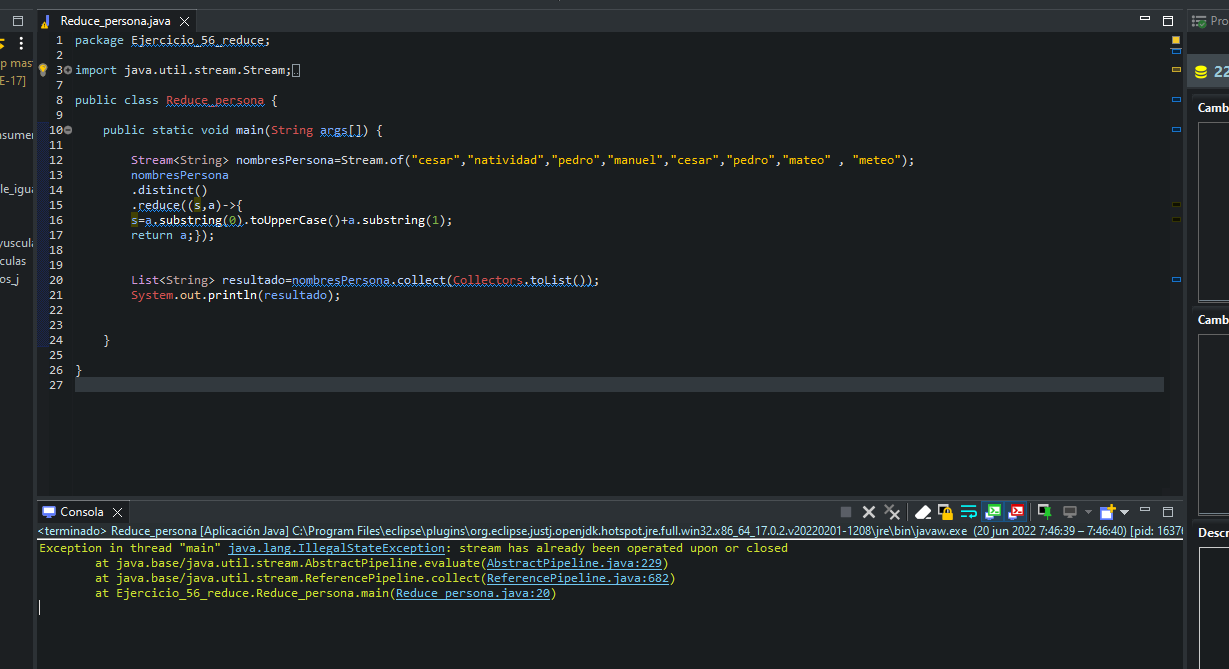
Filter(Predicate<T>:Stream<T> devuelve solo el Stream que contiene los elementos que cumplen el predicado en este caso startsWith(“J”);

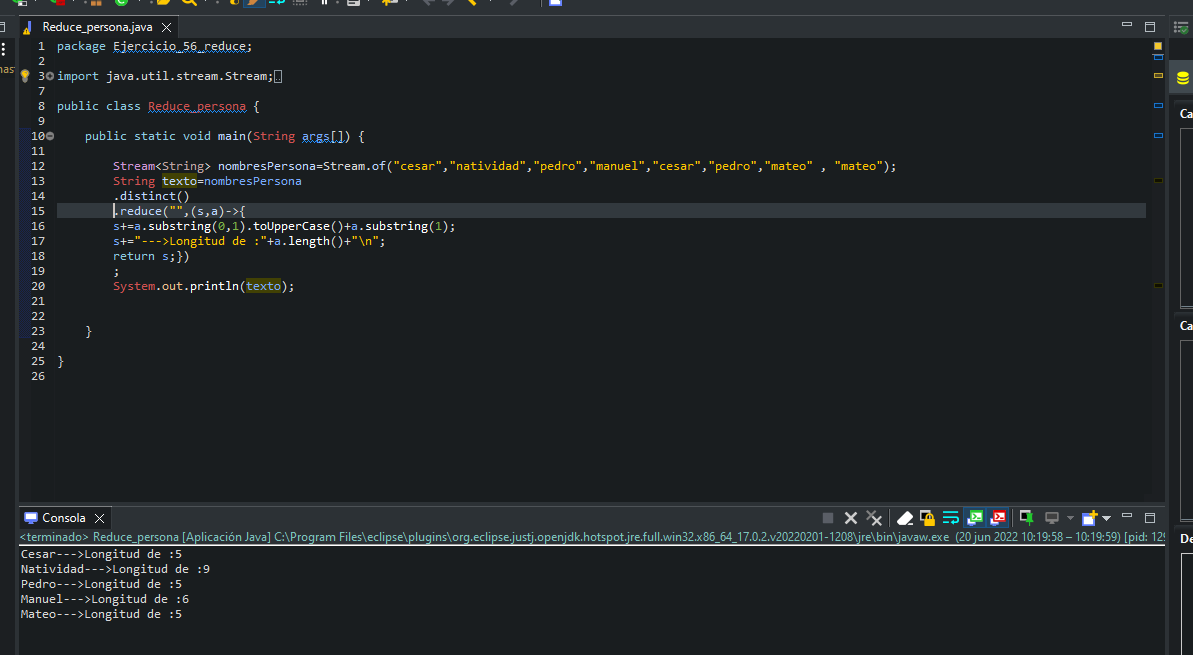
## Ejercicio 55: Crea un Stream de nombre de personas en mayúsculas, conviértelo en nombres con solo la primera en mayúscula (el resto en minúsculas), y usa las siguientes operaciones terminales (puedes inventarte tu la condición… que empiece por L, que acabe en “o”,…):  anyMatch  noneMatch  allMatch



CERRAR el Strem mostrar el original no se pude , volver a usar un stream

## Crea un Stream de nombre de personas en minúscula, usa la operación distinct() y muéstralo por pantalla el nombre con la primera en mayúsculas  Añádele la operación reduce()

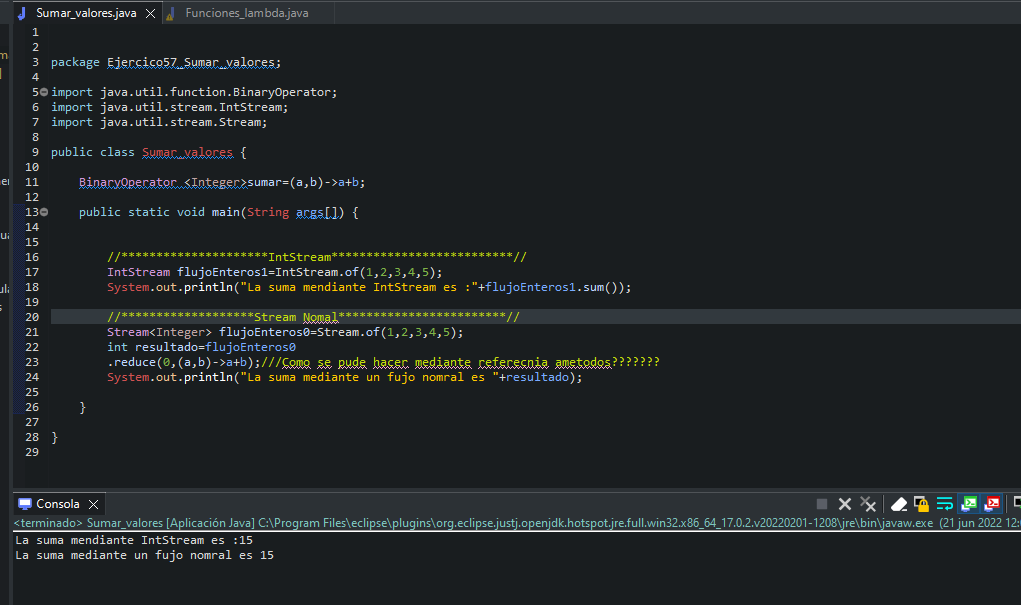




## Ejercicio 57

Desde java 8 se ofrece la posibilidad de crear flujos a partir de tres tipos primitivos: int,Long y doublé.Como Strean<T> es una interfaz genérica y no hay forma de usar primitivas como parámetro de tipo genérico , se crearon tres tipos de interfaces especiales :IntStrema,LongStream, DoubleStream

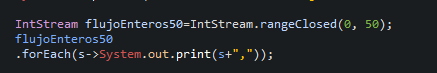
### Crea un Stream de int y añádele una operación que sume sus valores.



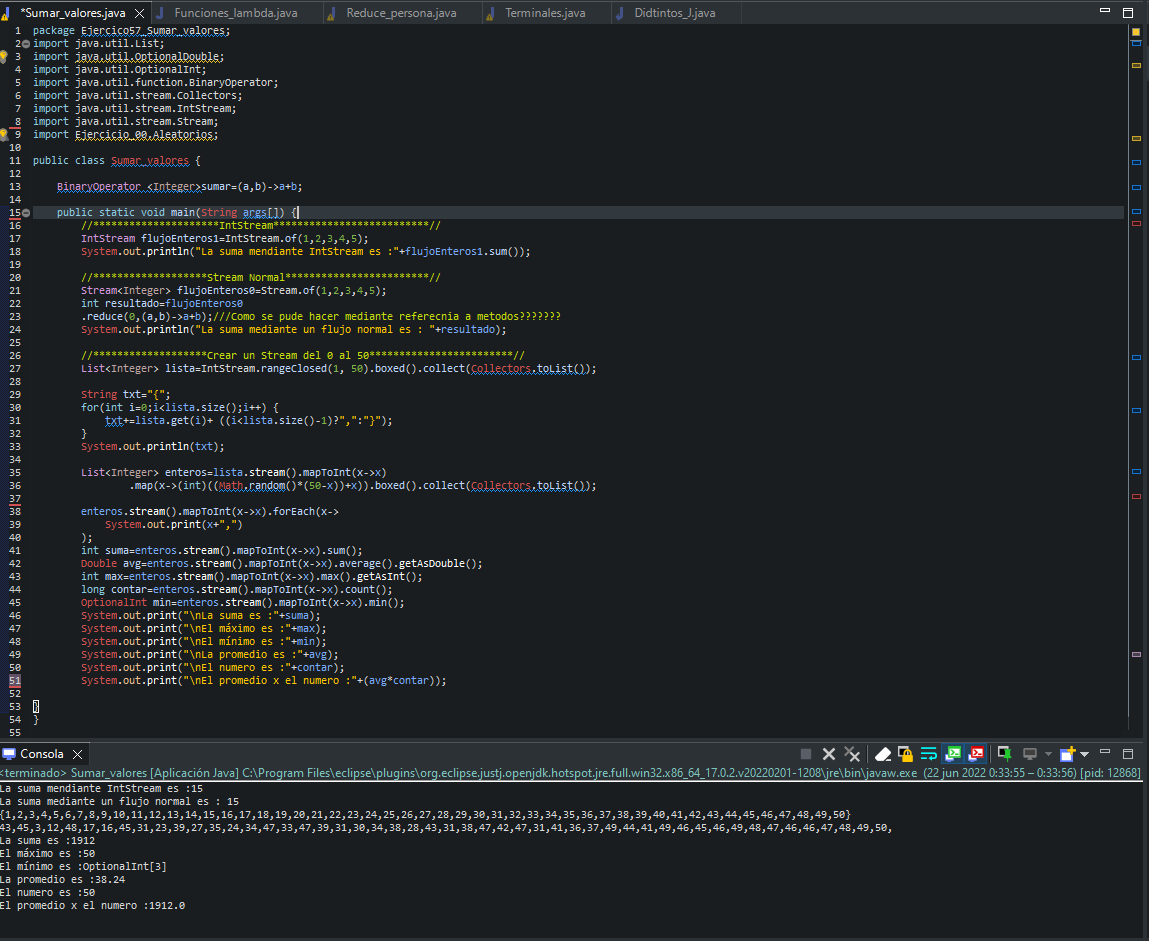
### Crea un IntStream con números entre el 1 y el 50 .

El método **range(int startInclusive, int endExclusive)** crea un flujo ordenado desde el primer parámetro hasta el segundo parámetro. Incrementa el valor de los elementos posteriores con el paso igual a 1. El resultado no incluye el último parámetro, es solo un límite superior de la secuencia.

El  ***método rangeClosed(int startInclusive, int endInclusive)***  hace lo mismo con solo una diferencia, se incluye el segundo elemento. Podemos usar estos dos métodos para generar cualquiera de los tres tipos de flujos de primitivas.

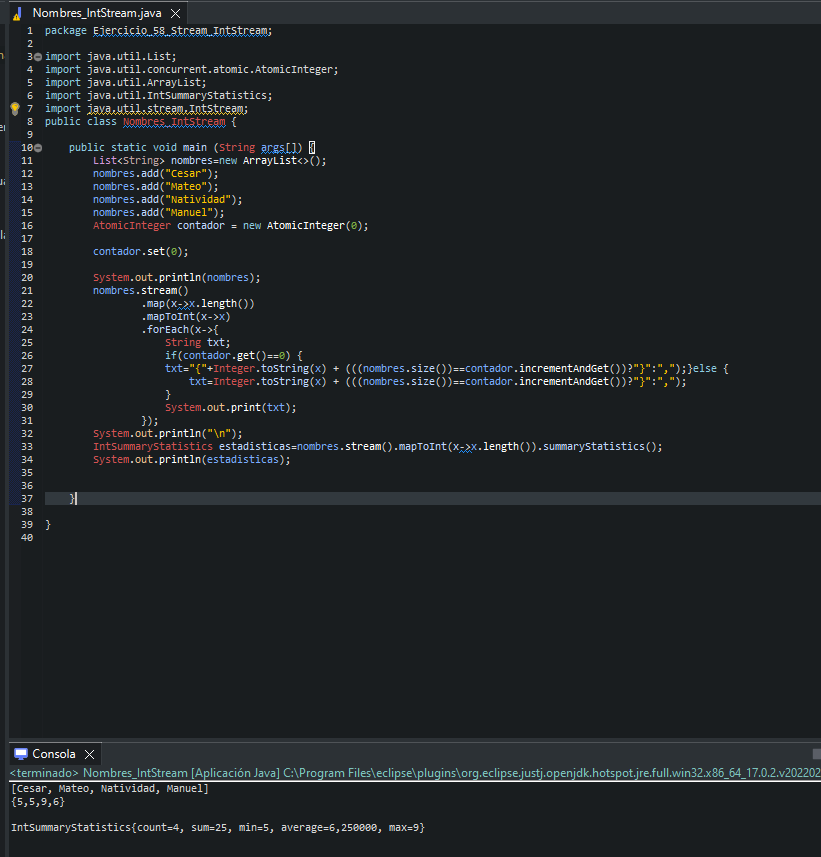


### A partir de IntStream anterior crea un IntStream con número aleatorios entre el valor y el valor + 50. Con el IntStream resultado, muestra por pantalla algunas de sus estadísticas (máximo valor, mínimo, sumatorio, contar, media,….) .



Buscar OptionalInt, Boxed()

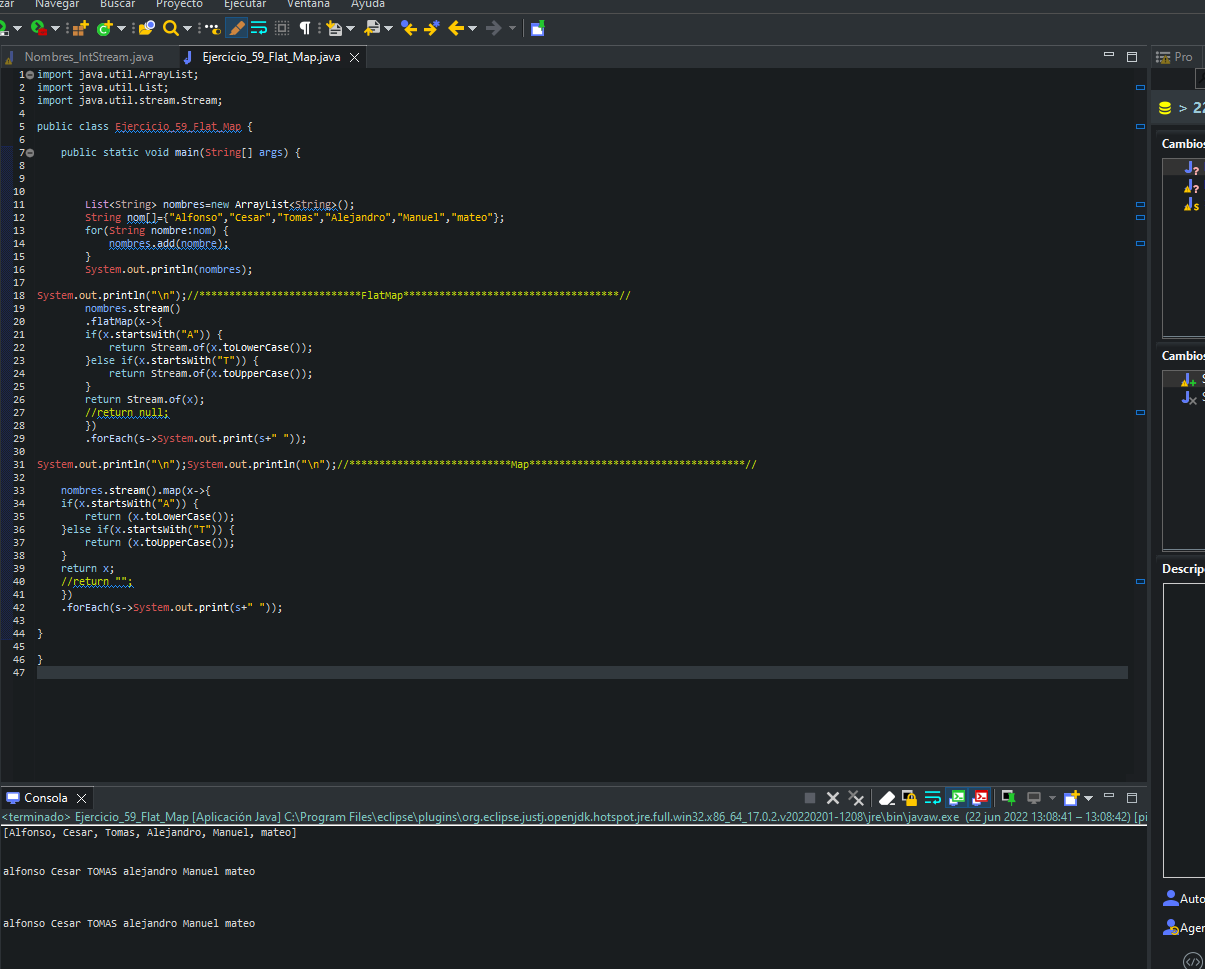
## Crea un Stream de nombres y obtén un IntStream con la longitud de cada uno de ellos.Crea un IntSummaryStatistics para obtener las estadística (máximo valor, mínimo, sumatorio, contar, media,….)



## Ejercicio 59: Con un Stream de nombres y usando la operación flatMap devuelve el nombre en mayúsculas si el nombre empieza por A y devuelve el nombre en minúsculas si empieza por T (por ejemplo, lo puedes adaptar a tu ejemplo)

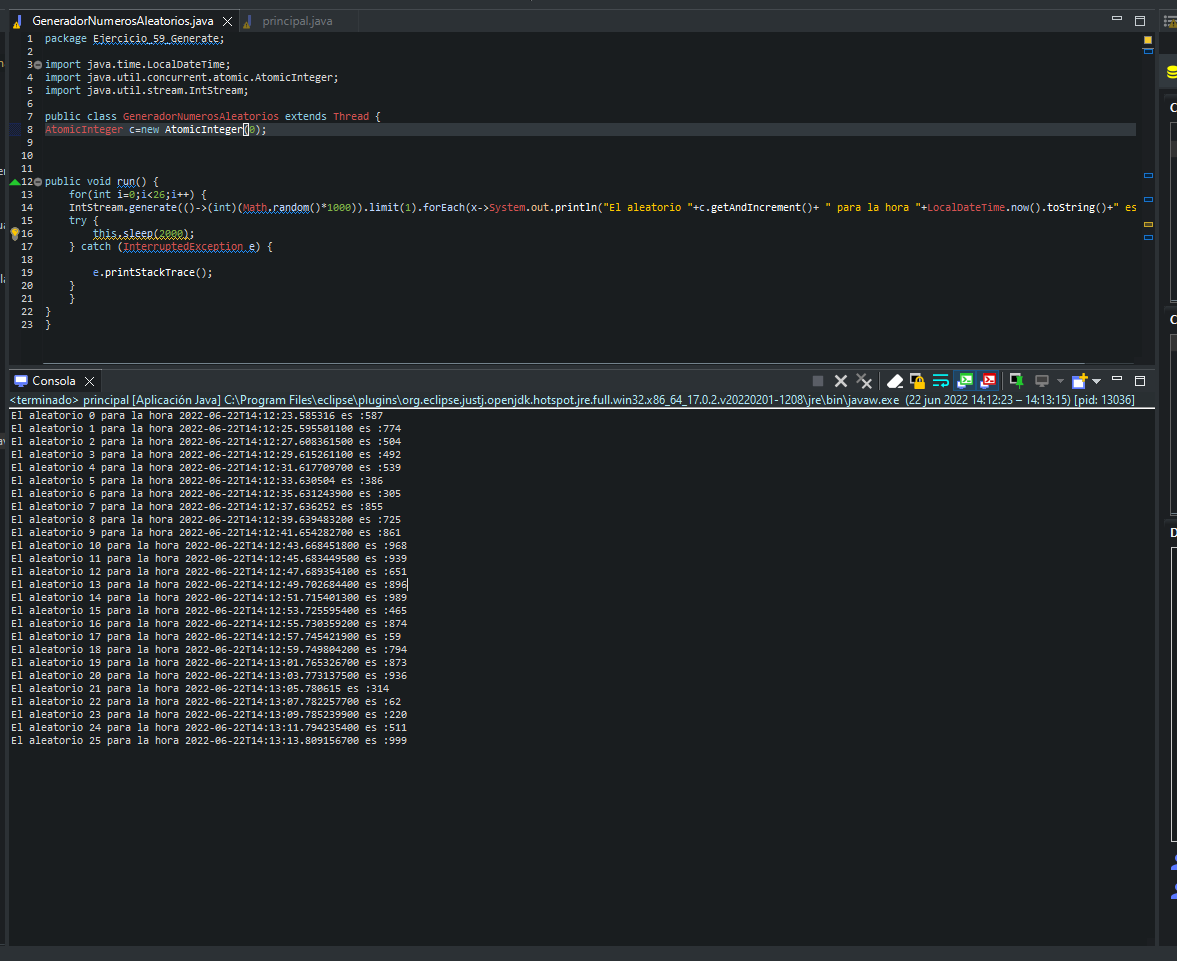
La diferencia es que map() devuelve el mismo número de elementos que el Stream de entrada ya que es simplemente una proyección de los elementos de entrada. Es decir cada elemento de entrada se transforma en un elemento de salida.

Por otro lado .flatMap(), proyecta una lista de elementos de cada elemento original y los concatena en un único stream.



En este ejemplo resulta inútil flatMap????

## Usa la función generate() para mostrar números aleatorios por pantalla cada X minisegundos (se duerme el hilo durante ese tiempo), con un límite de 25 números



## Usa la función generate() para mostrar números aleatorios por pantalla cada X minisegundos (muestra este tiempo también por pantalla), con un límite de 25 números. Haz lo mismo que el punto anterior pero usando la función parallel().

