LAMBDA

- **parametros: son los parámetros de la función.**
- **->: es el operador lambda que separa los parámetros del cuerpo de la función.**
- **{ cuerpo de la función }: es el código que se ejecutará cuando se invoque la función.**

(parametros) -> { cuerpo de la función }

FLTRAR NÚMEROS PARES DE UNA LISTA:

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6);
List<Integer> evenNumbers = numbers.stream()
                        .filter(n -> n % 2 == 0)
                        .collect(Collectors.toList());
System.out.println(evenNumbers);
// Resultado: [2, 4, 6]
```

filter para filtrar los números pares de la lista original.

clase anónima

```
public static void main(String[] args) {
    List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6);
    List<Integer> evenNumbers = numbers.stream()
                        .filter(new Predicate<Integer>() {
                            @Override
                            public boolean test(Integer n) {
                                return n % 2 == 0;
                            }
                        })
                        .collect(Collectors.toList());
    System.out.println(evenNumbers);
}
```

LEVAR AL CUADRADO CADA ELEMENTO DE UNA LISTA:

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
List<Integer> squaredNumbers = numbers.stream()
                        .map(n -> n * n)
                        .collect(Collectors.toList());

System.out.println(squaredNumbers);

// Resultado: [1, 4, 9, 16, 25]
```

map para calcular el cuadrado de cada elemento.

clase que implemente la interfaz Function.

```
public class Main {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
        List<Integer> squaredNumbers = numbers.stream()
                                .map(new SquareFunction())
                                .collect(Collectors.toList());
        System.out.println(squaredNumbers);
    }
}
//  interfaz Function para calcular el cuadrado de un número
class SquareFunction implements Function<Integer, Integer> {
    @Override
    public Integer apply(Integer n) {
        return n * n;
    }
}
```

ORDENAR UNA LISTA DE CADENAS POR LONGITUD:

```
List<String> names = Arrays.asList("Ana", "Carlos", "Eva", "Pedro",
"Juan");
names.sort((a, b) -> a.length() - b.length());
// Resultado: [Eva, Ana, Juan, Pedro, Carlos]
```

sort para ordenar la lista de cadenas según su longitud.

clase que implemente la interfaz Comparator.

```
public class Main {
    public static void main(String[] args) {
        List<String> names = Arrays.asList("Ana", "Carlos", "Eva", "Pedro", "Juan");
        names.sort(new LengthComparator());
        System.out.println(names);
    }
}
// Clase implementa la interfaz Comparator para comparar las longitudes de las
cadenas
class LengthComparator implements Comparator<String> {
    @Override
    public int compare(String a, String b) {
        return a.length() - b.length();
    }
}
```

VERIFICAR SI UNA LISTA CONTIENE UN ELEMENTO ESPECÍFICO:

```
List<String> fruits = Arrays.asList("manzana", "pera", "naranja", "plátano");
boolean containsOrange = fruits.stream()
                .anyMatch(fruit -> fruit.equals("naranja"));
// Resultado: true
```

anyMatch para verificar si la lista contiene el elemento "naranja".

el método contains de la clase List.

```
public class Main {
    public static void main(String[] args) {
        List<String> fruits = Arrays.asList("manzana", "pera", "naranja", "plátano");
        boolean containsOrange = containsFruit(fruits, "naranja");
        System.out.println(containsOrange);
    }
    public static boolean containsFruit(List<String> fruits, String targetFruit) {
        for (String fruit : fruits) {
            if (fruit.equals(targetFruit)) {
                return true;
            }
        }
        return false;
    }
}
```

CALCULAR LA SUMA DE UNA LISTA DE NÚMEROS:

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
int sum = numbers.stream().reduce(0, (a, b) -> a + b);
// Resultado: 15
```

reduce para calcular la suma de los elementos de la lista.

usando el método reduce con un BinaryOperator

```
public class Main {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
        int sum = calculateSum(numbers);
        System.out.println(sum);
    }
    public static int calculateSum(List<Integer> numbers) {
        BinaryOperator<Integer> sumOperator = new SumOperator();
        return numbers.stream().reduce(0, sumOperator);
    }
}
// Clase que implementa BinaryOperator para sumar dos números enteros
class SumOperator implements BinaryOperator<Integer> {
    @Override
    public Integer apply(Integer a, Integer b) {
        return a + b;
    }
}
```

CONTAR EL NÚMERO DE CADENAS QUE TIENEN MÁS DE 5 CARACTERES:

```
List<String> words = Arrays.asList("casa", "coche", "perro", "árbol", "sol");
long count = words.stream().filter(word -> word.length() > 5).count()
// Resultado: 2
```

filter para seleccionar las cadenas con más de 5 caracteres.

```
public class Main {
    public static void main(String[] args) {
        List<String> words = Arrays.asList("casa", "coche", "perro", "árbol", "sol");
        long count = countWordsLongerThan5(words);
        System.out.println(count);
    }
    public static long countWordsLongerThan5(List<String> words) {
        long count = 0;
        for (String word : words) {
            if (word.length() > 5) {
                count++;
            }
        }
        return count;
    }
}
```

CONVERTIR UNA LISTA DE CADENAS A MAYÚSCULAS:

```
List<String> fruits = Arrays.asList("manzana", "pera", "naranja");
List<String> uppercaseFruits = fruits.stream()
                    .map(fruit -> fruit.toUpperCase())
                    .collect(Collectors.toList());
// Resultado: [MANZANA, PERA, NARANJA]
```

map para convertir cada cadena a mayúsculas.

uso de un Function personalizado

```
public class Main {
    public static void main(String[] args) {
        List<String> fruits = Arrays.asList("manzana", "pera", "naranja");
        List<String> uppercaseFruits = convertToUppercase(fruits);

        System.out.println(uppercaseFruits);
    }

    public static List<String> convertToUppercase(List<String> fruits) {
        Function<String, String> uppercaseFunction = new UppercaseConverter();
        return fruits.stream()
                .map(uppercaseFunction)
                .collect(Collectors.toList());
    }
}

// Clase que implementa Function para convertir una cadena a mayúsculas
class UppercaseConverter implements Function<String, String> {
    @Override
    public String apply(String fruit) {
        return fruit.toUpperCase();
    }
}
```

ENCONTRAR EL MÁXIMO DE UNA LISTA DE NÚMEROS:

```
List<Integer> numbers = Arrays.asList(15, 7, 21, 13, 9);
int max = numbers.stream().max((a, b) -> a.compareTo(b)).orElse(0);
// Resultado: 21
```

max para encontrar el valor máximo de la lista

método Collections.max() proporcionando un comparador personalizado que implemente la interfaz Comparator

```
public class Main {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(15, 7, 21, 13, 9);
        int max = findMax(numbers);

        System.out.println(max);
    }

    public static int findMax(List<Integer> numbers) {
        Comparator<Integer> comparator = new MaxComparator();
        return Collections.max(numbers, comparator);
    }
}

// Clase que implementa Comparator para encontrar el máximo
class MaxComparator implements Comparator<Integer> {
    @Override
    public int compare(Integer a, Integer b) {
        return a.compareTo(b);
    }
}
```

ENCONTRAR EL MÍNIMO DE UNA LISTA DE NÚMEROS:

```
List<Integer> numbers = Arrays.asList(15, 7, 21, 13, 9)

int min = numbers.stream().min((a, b) -> a.compareTo(b)).orElse(0)
// Resultado: 7
```

min para encontrar el valor mínimo de la lista de números.

método Collections.min() proporcionando un comparador personalizado que implemente la interfaz Comparator

```
public class Main {
  public static void main(String[] args) {
    List<Integer> numbers = Arrays.asList(15, 7, 21, 13, 9);
    int min = findMin(numbers);
    System.out.println(min);
  }
  public static int findMin(List<Integer> numbers) {
    Comparator<Integer> comparator = new MinComparator();
    return Collections.min(numbers, comparator);
  }
}
// Clase que implementa Comparator para encontrar el mínimo
class MinComparator implements Comparator<Integer> {
  @Override
  public int compare(Integer a, Integer b) {
    return a.compareTo(b);
  }
}
```

VERIFICAR SI TODOS LOS ELEMENTOS DE LA LISTA SON IMPARES,

```
List<Integer> numbers = Arrays.asList(1, 3, 5, 7, 9)
boolean allOdd = numbers.stream().noneMatch(n -> n % 2 == 0)

// Resultado: True
```

noneMatch devuelve true si ninguno de los elementos de la lista cumple con la condición especificada en la lambda. En este caso, la condición evaluaría si un número es impar, es decir, si su residuo al dividirlo por 2 es diferente de cero.

```
public class Main {
  public static void main(String[] args) {
    List<Integer> numbers = Arrays.asList(1, 3, 5, 7, 9);
    boolean allOdd = areAllNumbersOdd(numbers);
    System.out.println(allOdd);
  }
  public static boolean areAllNumbersOdd(List<Integer> numbers) {
    for (Integer number : numbers) {
      if (number % 2 == 0) {
        return false;
      }
    }
    return true;
  }
}
```