La programación funcional es un paradigma de programación que se centra en el uso de funciones puras y la evitación de efectos secundarios. En Java, la programación funcional se puede lograr utilizando características como expresiones lambda.

**Lista de programacion funcional de java**

Aquí hay algunas características de programación funcional que se pueden encontrar en Java:

1. **Expresiones lambda**: Las expresiones lambda son funciones anónimas que se pueden pasar como argumentos o asignar a variables. Se utilizan para expresar código de manera más concisa y permiten el uso de funciones como argumentos.
2. **Interfaces funcionales**: Las interfaces funcionales son interfaces que tienen un solo método abstracto y se utilizan para definir el tipo de una expresión **lambda**.
3. **Streams**: Los **streams** son secuencias de elementos que se pueden procesar de manera funcional. Los **streams** se utilizan para representar datos de manera abstracta y se pueden manipular utilizando operaciones de **filtrado, mapeo, ordenación y reducción**.
4. **Funciones de alto orden**: Las funciones de alto orden son funciones que toman una o más funciones como argumentos o devuelven una función como resultado. Se utilizan para crear abstracciones y reducir la duplicación de código.
5. **Composición de funciones**: La composición de funciones es una técnica que se utiliza para combinar dos o más funciones en una sola función. Esto se logra aplicando una función a la salida de otra función.
6. **Inmutabilidad**: La inmutabilidad se refiere a la incapacidad de un objeto para cambiar una vez que se ha creado. En Java, se pueden crear objetos inmutables utilizando clases final, campos final o clases de valor.
7. **Recursión**: La recursión es una técnica que se utiliza para resolver problemas al descomponerlos en subproblemas más pequeños. En Java, se pueden crear funciones recursivas para solucionar problemas de manera funcional.

**Ejemplos de Expresiones lambda**

1. Expresión lambda para una función que suma dos números:

int resultado = ((a, b) -> a + b).apply(2, 3);

1. Expresión lambda para una función que filtra los números pares en una lista:

List<Integer> numeros = Arrays.asList(1, 2, 3, 4, 5);

List<Integer> pares = numeros.stream()

.filter(n -> n % 2 == 0)

.collect(Collectors.toList());

1. Expresión lambda para una función que ordena una lista de cadenas: