LAMBDA EXPRESSION & FUNCTIONAL INTERFACE

A lambda function is a way to represent an anonymous function.
Is a block of code that can be passed around and executed later.

Syntax:
(parameters) -> expression | (parameters) -> { //mutiline }
Example:
```
Runnable r=(s) -> System.out.println (s)
```

METHOD REFERENCES
Is a shorthand syntax for a lambda that simply calls an existing method.
Syntax: ClassName : methodName
Example:
```
list.ForEach (System.out::println)
```
Reference to a constructor
```
ArrayList::new | () -> new ARRAYLIST <>()
```

FUNCTIONAL INTERFACE JAVA.UTIL.FUNCTION
Is an interface with one abstract method, it serves as target for lambda expressions and method references enabling behavior to be passed as parameters. java.util.Function
Example

INTERFACE	ABSTRACT METHOD	DESCRIPTION
Predicate<T>	boolean test<T t>	Test a condition on a single input
Function<T,R>	R apply (T t)	Transforms a value from T to R
Consumer<T>	void accept (T t)	Performs an action with no return
Supplier<T>	T get ()	Provides a value without input
UnaryOperator<T>	T apply (T t)	A function from T to T

Example:

```
// Bi predicate: Check if two strings are equal
BiPredicate<String, String> equalsIgnoreCase =
        (a, b) -> a.equalsIgnoreCase(b);
```

Custom Functional Interface

```
@FunctionalInterface
interface MathOperation {
    int operate(int a, int b);
}
MathOperation add = (a, b) -> a + b;

System.out.println(add.operate(5, 3));
```