Types	
PREDICATE	takes one (or two) argument(s) and returns a boolean
UNARY OPERATOR	result and the single argument types are the same
BINARY OPERATOR	result and both argument types are the same
FUNCTION	result and one (or two) argument(s) types are different
SUPPLIER	takes no arguments, returns a value
CONSUMER	takes one (or two) arguments and returns no value

Notes
If the interface accepts primitive arguments: prefixed Double, Int, Long, e.g. DoubleConsumer
If the interface produces a primitive result: prefixed ToDouble, ToInt, ToLong, e.g. ToDoubleFunction
If the interface both accepts and produces a primitive: prefixes combined e.g. IntToDoubleFunction
BiConsumer variants that accept an object type and a primitive are prefixed Obj + the primitive, e.g. ObjDoubleConsumer

Predicate		
Predicate<T>	boolean test(T t)	Represents a predicate (boolean-valued function) of one argument (reference type)
BiPredicate<T,U>	boolean test(T t, U u)	Accepts two arguments (reference types)
DoublePredicate	e boolean test(double value)	Accepts one double-valued argument
IntPredicate	boolean test(int value)	Accepts one int-valued argument
LongPredicate	boolean test(long value)	Accepts one long-valued argument

Supplier		
Supplier<T>	T get()	Represents a supplier of results (reference type)
DoubleSupplier	double getAsDouble()	A supplier of double-valued results
IntSupplier	int getAsInt()	A supplier of int-valued results
LongSupplier	long getAsLong()	A supplier of long-valued results
BooleanSupplier	boolean getAsBoolean()	A supplier of boolean-valued results

Consumer		
Consumer<T>	void accept(T t)	Represents an operation that accepts a single (reference type) input argument and returns no result
DoubleConsumer	void accept(double value)	Accepts a single double-valued argument and returns no result
IntConsumer	void accept(int value)	Accepts a single int-valued argument and returns no result
LongConsumer	void accept(long value)	Accepts a single long-valued argument and returns no result
BiConsumer<T,U>	void accept(T t, U u)	Represents an operation that accepts two (reference type) input arguments and returns no result
ObjDoubleConsumer<T>	void accept(T t, double value)	Accepts an object-valued and a double-valued argument, and returns no result

Consumer (cont)		
ObjIntConsumer<T>	void accept(T t, int value)	Accepts an object-valued and an int-valued argument, and returns no result
ObjLongConsumer<T>	void accept(T t, long value)	Accepts an object-valued and a long-valued argument, and returns no result

Binary Operator		
BinaryOperator<T>	T apply(T t, T u)	Represents an operation upon two operands of the same type, producing a result of the same type as the operands (reference type)
DoubleBinaryOperator	double applyAsDouble(double left, double right)	Accepts two double-valued operands and produces a double-valued result
IntBinaryOperator	int applyAsInt(int left, int right)	Accepts two int-valued operands and produces an int-valued result
LongBinaryOperator	long applyAsLong(long left, long right)	Accepts two long-valued operands and produces a long-valued result.

Unary Operator		
UnaryOperator<T>	T apply(T t)	Represents an operation on a single operand that produces a result of the same type as its operand (reference type)
DoubleUnaryOperator	double applyAsDouble(double operand)	Accepts single double-valued operand and produces a double-valued result
IntUnaryOperator	int applyAsInt(int operand)	Accepts a single int-valued operand and produces an int-valued result
LongUnaryOperator	long applyAsLong(long operand)	Accepts a single long-valued operand and produces a long-valued result

Function		
Function<T,R>	R apply(T t)	Represents a function that accepts one argument and produces a result (reference type)
DoubleFunction<R>	R apply(double value)	Accepts a double-valued argument and produces a result
IntFunction<R>	R apply(int value)	Accepts an int-valued argument and produces a result
LongFunction<R>	R apply(long value)	Accepts a long-valued argument and produces a result
DoubleToIntFunction	int applyAsInt(double value)	Accepts a double-valued argument and produces an int-valued result
DoubleToLongFunction	long applyAsLong(double value)	Accepts a double-valued argument and produces a long-valued result
IntToDoubleFunction	double applyAsDouble(int value)	Accepts an int-valued argument and produces a double-valued result

Function (cont)

IntToLongFunction	long applyAsLong(int value)	Accepts an int-valued argument and produces a long-valued result
LongToIntFunction	int applyAsInt(long value)	Accepts a long-valued argument and produces an int-valued result
LongToDoubleFunction	double applyAsDouble(long value)	Accepts a long-valued argument and produces a double-valued result.
ToDoubleFunction<T>	double applyAsDouble(T value)	Accepts a reference type and produces an int-valued result
ToIntFunction<T>	int applyAsInt(T value)	Accepts a reference type and produces an int-valued result
ToLongFunction<T>	long applyAsLong(T value)	Accepts a reference type and produces a long-valued result.
BiFunction<T,U,R>	R apply(T t, U u)	Represents a function that accepts two arguments and produces a result (reference type)
ToDoubleBiFunction-<T,U>	double applyAsDouble(T t, U u)	Accepts two reference type arguments and produces a double-valued result
ToIntBiFunction<T,U>	int applyAsInt(T t, U u)	Accepts two reference type arguments and produces an int-valued result
ToLongBiFunctio-n<T,U>	long applyAsLong(T t, U u)	Accepts two reference type arguments and produces a long-valued result

Java Functional Interface

The functional interface is a simple interface with only one abstract method. A lambda expression can be used through a functional interface in Java 8. We can declare our own/custom functional interface by defining the Single Abstract Method (SAM) in an interface.

Custom Interface

```
@FunctionalInterface
interface Custom Fun cti ona lIn terface {
     void display();
}
```