Interfaces Funcionales en Java (Enfoque Funcional)

Interfaz Funcional	Tipo Entrada/Salida	Método Abstracto	Uso Típico
Predicate<T>	T → boolean	boolean test(T t)	Filtrado de datos basado en condiciones
Consumer<T>	T → void	void accept(T t)	Efectos colaterales con datos de tipo T
Function<T, R>	T → R	R apply(T t)	Transformación de datos de tipo T a tipo R
Supplier<T>	() → T	T get()	Generación de datos de tipo T
UnaryOperator<T>	T → T	T apply(T t)	Modificación de datos preservando el tipo
BinaryOperator<T>	(T, T) → T	T apply(T t1, T t2)	Combinación de dos datos de tipo T en uno
BiPredicate<T, U>	(T, U) → boolean	boolean test(T t, U u)	Validación de condiciones con dos datos
BiConsumer<T, U>	(T, U) → void	void accept(T t, U u)	Efectos colaterales con dos datos
BiFunction<T, U, R>	(T, U) → R	R apply(T t, U u)	Transformación de dos datos en uno de tipo R
Runnable	() → void	void run()	Ejecución de tareas sin parámetros ni retorno
IntConsumer	int → void	void accept(int value)	Consumir datos primitivos de tipo int
IntFunction<R>	int → R	R apply(int value)	Construir datos de tipo R desde un int
ToIntFunction<T>	T → int	int applyAsInt(T value)	Extraer datos int desde datos de tipo T
IntPredicate	int → boolean	boolean test(int value)	Validación de datos numéricos enteros
IntSupplier	() → int	int getAsInt()	Generación de datos de tipo int
IntUnaryOperator	int → int	int applyAsInt(int value)	Modificación de datos de tipo int
IntBinaryOperator	(int, int) → int	int applyAsInt(int a, int b)	Combinación de dos datos de tipo int
LongConsumer	long → void	void accept(long value)	Consumir datos primitivos de tipo long
DoubleConsumer	double → void	void accept(double value)	Consumir datos primitivos de tipo double
ToDoubleFunction<T>	T → double	double applyAsDouble(T value)	Extraer datos double desde datos de tipo T
DoubleToIntFunction	double → int	int applyAsInt(double value)	Transformación de datos double a int