Appendix

The standard functional interfaces

Consumer			
Synopsis	**Consume & discard**		
Functional interface	**Type**[1]	**Return**	**Method**
Consumer<T>	Abs	void	accept(T t);
	Def	Consumer<T>	andThen(Consumer<? super T> after);
BiConsumer<T, U>	Abs	void	accept(T t, U u);
	Def	BiConsumer<T, U>	andThen(BiConsumer<? super T, ? super U> after);
DoubleConsumer	Abs	void	accept(double value);
	Def	DoubleConsumer	andThen(DoubleConsumer after);
IntConsumer	Abs	void	accept(T t);
	Def	IntConsumer	andThen(IntConsumer after);
LongConsumer	Abs	void	accept(long value);
	Def	LongConsumer	andThen(LongConsumer after);
ObjDoubleConsumer<T>	Abs	void	accept(T t, double value);
ObjIntConsumer<T>	Abs	void	accept(T t, int value);
ObjLongConsumer<T>	Abs	void	accept(T t, long value);
Function			
Synopsis	**Transform/Compute**		
Functional interface	**Type**	**Return**	**Method**
Function<T, R>	Abs	R	apply(T t);
	Def	Function<V, R>	compose(Function<? super V, ? extends T> before);
	Def	Function<T, V>	andThen(Function<? super R, ? extends V> after);
	Stat	Function<T, T>	identity();
BiFunction<T, U, R>	Abs	R	apply(T t);
	Def	BiFunction<T, U, V>	andThen(Function<? super R, ? extends V> after);
BinaryOperator<T>	Stat	BinaryOperator<T>	minBy(Comparator<? super T> comparator);
	Stat	BinaryOperator<T>	maxBy(Comparator<? super T> comparator);
DoubleFunction<R>	Abs	R	apply(double value);
DoubleToIntFunction	Abs	int	applyAsInt(double value);
DoubleToLongFunction	Abs	long	applyAsLong(double value);
IntFunction<R>	Abs	R	apply(int value);
IntToDoubleFunction	Abs	double	applyAsDouble(int value);
IntToLongFunction	Abs	long	applyAsLong(int value);
LongFunction<R>	Abs	R	apply(long value);
LongToDoubleFunction	Abs	double	applyAsDouble(long value);
LongToIntFunction	Abs	int	applyAsInt(long value);
ToDoubleBiFunction<T, U>	Abs	double	applyAsDouble(T t, U u);
ToDoubleFunction<T>	Abs	double	applyAsDouble(T value);
ToIntBiFunction	Abs	int	applyAsInt(T t, U u);
ToIntFunction	Abs	int	applyAsInt(T value);

[1] **Abs**: Abstract method, **Def**: Default method, **Stat**: Static method

ToIntBiFunction<T, U>	Abs	long	applyAsLong(T t, U u);
ToLongFunction<T>	Abs	long	applyAsLong(T value);
DoubleBinaryOperator	Abs	double	applyAsDouble(double left, double right);
DoubleUnaryOperator	Abs	double	applyAsDouble(double operand);
	Def	DoubleUnaryOperator	compose(DoubleUnaryOperator before);
	Def	DoubleUnaryOperator	andThen(DoubleUnaryOperator after);
	Stat	DoubleUnaryOperator	identity();
IntBinaryOperator	Abs	int	applyAsInt(int left, int right);
IntUnaryOperator	Abs	int	applyAsInt(int operand);
	Def	IntUnaryOperator	compose(IntUnaryOperator before);
	Def	IntUnaryOperator	andThen(IntUnaryOperator after);
	Stat	IntUnaryOperator	identity();
LongBinaryOperator	Abs	long	applyAsLong(long left, long right);
LongUnaryOperator	Abs	long	applyAsLong(long operand);
	Def	LongUnaryOperator	compose(LongUnaryOperator before);
	Def	LongUnaryOperator	andThen(LongUnaryOperator after);
	Stat	LongUnaryOperator	identity();
UnaryOperator<T>	Stat	UnaryOperator<T>	identity();
Predicate			
Synopsis	**Test/Filter**		
Functional interface	**Type**	**Return**	**Method**
Predicate<T>	Abs	boolean	test(T t);
	Def	Predicate<T>	and(Predicate<? super T> other);
	Def	Predicate<T>	negate();
	Def	Predicate<T>	or(Predicate<? super T> other);
	Stat	Predicate<T>	isEqual(Object targetRef);
BiPredicate<T, U>	Abs	boolean	test(T t);
	Def	BiPredicate<T, U>	and(BiPredicate<? super T, ? super U> other);
	Def	BiPredicate<T, U>	negate();
	Def	BiPredicate<T, U>	or(BiPredicate<? super T, ? super U> other);
DoublePredicate	Abs	boolean	test(double value);
	Def	DoublePredicate	and(DoublePredicate other);
	Def	DoublePredicate	negate();
	Def	DoublePredicate	or(DoublePredicate other);
IntPredicate	Abs	boolean	test(int value);
	Def	IntPredicate	and(IntPredicate other);
	Def	IntPredicate	negate();
	Def	IntPredicate	or(IntPredicate other);
LongPredicate	Abs	boolean	test(long value);
	Def	LongPredicate	and(LongPredicate other);
	Def	LongPredicate	negate();
	Def	LongPredicate	or(LongPredicate other);
Supplier			
Synopsis	**Create**		
Functional interface	**Type**	**Return**	**Method**
Supplier	Def	T	get();
BooleanSupplier	Def	boolean	getAsBoolean()
DoubleSupplier	Def	double	getAsDouble();
IntSupplier	Def	int	getAsInt();

LongSupplier	Def	long	getAsLong();
Comparator			
Description	**Test two objects for equivalence**		
Functional interface	**Type**	**Return**	**Method**
Comparator<T>	Abs	int	compare(T o1, T o2);
	Def	Comparator<T>	reversed();
	Def	Comparator<T>	thenComparing(Comparator<? super T> other);
	Def	Comparator<T>	thenComparing(Function<? super T, ? extends U> keyExtractor, Comparator<? super U> keyComparator);
	Def	Comparator<T>	thenComparing(Function<? super T, ? extends U> keyExtractor);
	Def	Comparator<T>	thenComparingInt(ToIntFunction<? super T> keyExtractor);
	Def	Comparator<T>	thenComparingLong(ToLongFunction<? super T> keyExtractor);
	Def	Comparator<T>	thenComparingDouble(ToDoubleFunction<? super T> keyExtractor);
	Stat	Comparator<T>	reverseOrder();
	Stat	Comparator<T>	naturalOrder();
	Stat	Comparator<T>	nullsFirst(Comparator<? super T> comparator;
	Stat	Comparator<T>	nullsLast(Comparator<? super T> comparator);
	Stat	Comparator<T>	comparing(Function<? super T, ? extends U> keyExtractor,Comparator<? super U> keyComparator);
	Stat	Comparator<T>	comparing(Function<? super T, ? extends U> keyExtractor);
	Stat	Comparator<T>	comparingInt(ToIntFunction<? super T> keyExtractor);
	Stat	Comparator<T>	comparingLong(ToLongFunction<? super T> keyExtractor);
	Stat	Comparator<T>	comparingDouble(ToDoubleFunction<? super T> keyExtractor);

The Stream Interface

Build				
Synopsis	**Create a stream**			
Variants	**IntStream, LongStream, DoubleStream**			
Return	**Method**	**Cont**[2]	**Type**[3]	**Synopsis**
Stream<T>	concat(Stream<? extends T> a, Stream<? extends T> b)	Intr	Stat	Concatenates two streams to form a new one.
Stream<T>	empty()	Intr	Stat	Creates an empty stream.
Stream<T>	generate(Supplier<T> s)	Intr	Stat	Creates a stream based on the given *supplier.*
Stream<T>	iterate(final T seed, final UnaryOperator<T> f)	Intr	Stat	Iterate through a stream from a starting point using *f* to generate the next element.
Stream<T>	of(T t)	Intr	Stat	Creates a stream of one element of type *T.*
Stream<T>	of(T... values)	Intr	Stat	Creates a stream of one or more element of type *T.*
S	onClose(Runnable closeHandler)	Intr	Inst	Returns an equivalent stream with an additional *closeHandler.*
S	parallel()	Intr	Inst	Returns a parallel representation of the stream.
S	sequential()	Intr	Inst	Returns a sequential representation of the stream.
Stream<T>	skip(long n)	Intr	Inst	Discard the first *n* elements and returns the remainder of the stream.
Stream<T>	sorted()	Intr	Inst	Sorts the stream based on its natural order.
Stream<T>	sorted(Comparator<? super T> comparator)	Intr	Inst	Sorts the stream based on the given comparator.
S	unordered()	Intr	Inst	Returns an unordered stream
Builder<T>	builder()	Term	Stat	Returns a builder allowing the stream to be mutated.
void	close()	Term	Inst	Closes this stream, causing all close handlers for this stream pipeline to be called.
Iterator<T>	iterator()	Term	Inst	Returns an *iterator.*
Spliterator<T>	spliterator()	Term	Inst	Returns a *spliterator.*
Object[]	toArray()	Term	Inst	Converts the stream into an array of O*bject* objects.
A[]	toArray(IntFunction<A[]> generator)	Term	Inst	Converts the stream into an array of *A* based on an *IntFunction.*

[2] Continuity: **Intr**: Intermediate, **Term**: Terminal
[3] Type: **Inst**: Instance, **Stat**: Static

		Iterate		
Synopsis	**Traverse a stream**			
Return	**Method**	**Cont**	**Type**	**Synopsis**
void	forEach(Consumer<? super T> action)	Term	Inst	Iterates through the stream applying the *Consumer* to each element.
void	forEachOrdered(Consumer<? super T> action)	Term	Inst	Iterates through the stream applying the *Consumer* function to each element and guaranteeing the order of the stream if the stream has one.
		Filter		
Synopsis	**Filter stream elements**			
Return	**Method**	**Cont**	**Type**	**Synopsis**
Stream<T>	distinct()	Intr	Inst	Filters out duplicate elements and creates a new stream.
Stream<T>	filter(Predicate<? super T> predicate)	Intr	Inst	Filters elements of the stream based on the predicate condition.
		Map		
Synopsis	**Transform elements of the stream**			
Return	**Method**	**Cont**	**Type**	**Synopsis**
Stream<R>	flatMap(Function<? super T, ? extends Stream<? extends R>> mapper)	Intr	Inst	Returns a *Stream* consisting of the results of replacing each element of this stream with the contents of the stream produced by applying the provided mapping function to each element.
DoubleStream	flatMapToDouble(Function<? super T, ? extends DoubleStream> mapper)	Intr	Inst	Returns a *DoubleStream* consisting of the results of replacing each element of this stream with the contents of the stream produced by applying the provided mapping function to each element.
IntStream	flatMapToInt(Function<? super T, ? extends IntStream> mapper)	Intr	Inst	Returns an *IntStream* consisting of the results of replacing each element of this stream with the contents of the stream produced by applying the provided mapping function to each element.
LongStream	flatMapToLong(Function<? super T, ? extends LongStream>	Intr	Inst	Returns a *LongStream* consisting of the results of replacing each element

	mapper)			of this stream with the contents of the stream produced by applying the provided mapping function to each element.
Stream<R>	map(Function<? super T, ? extends R> mapper)	Intr	Inst	Returns a *Stream* consisting of the results of applying the given function to the elements of this stream.
DoubleStream	mapToDouble(ToDoubleFunction<? super T> mapper)	Intr	Inst	Returns a *DoubleStream* consisting of the results of applying the given function to the elements of this stream.
IntStream	mapToInt(ToIntFunction<? super T> mapper)	Intr	Inst	Returns an *IntStream* consisting of the results of applying the given function to the elements of this stream.
LongStream	mapToLong(ToLongFunction<? super T> mapper)	Intr	Inst	Returns a *LongStream* consisting of the results of applying the given function to the elements of this stream.
		Reduce		
Synopsis	**Reduce the stream to a value**			
Return	**Method**	**Cont**	**Type**	**Synopsis**
boolean	allMatch(Predicate<? super T> predicate)	Term	Inst	Returns true if all elements in the stream match the *Predicate* condition.
boolean	anyMatch(Predicate<? super T> predicate)	Term	Inst	Returns true if at least one element in the stream matches the *Predicate* condition.
R	collect(Collector<? super T, A, R> collector)	Term	Inst	Aggregates the stream using the collector providing functions for each step of the collection process.
R	collect(Supplier<R> supplier, BiConsumer<R, ? super T> accumulator, BiConsumer<R, R> combiner)	Term	Inst	Aggregates the stream using a *Supplier*, *Accumulator,* and *Combiner.*
long	count()	Term	Inst	Counts the elements in the stream.

Optional<T>	findAny()	Term	Inst	Returns any element in the stream as an *Optional*. Does not necessarily return the first element.
Optional<T>	findFirst()	Term	Inst	Returns the first element of the stream as an *Optional*.
Optional<T>	max(Comparator<? super T> comparator)	Term	Inst	Returns the maximum value in the stream as defined by the *Comparator.*
Optional<T>	min(Comparator<? super T> comparator)	Term	Inst	Returns the minimum value in the stream as defined by the *Comparator.*
boolean	noneMatch(Predicate<? super T> predicate)	Term	Inst	Returns true if no element in the stream matches the *Predicate* condition.
Optional<T>	reduce(BinaryOperator<T> accumulator)	Term	Inst	Aggregates the stream as one value.
T	reduce(T identity, BinaryOperator<T> accumulator)	Term	Inst	Aggregates the stream as one value using *T* as the starting value.
U	reduce(U identity, BiFunction<U, ? super T, U> accumulator, BinaryOperator<U> combiner)	Term	Inst	Aggregates the stream as one value using *T* as the starting value and a combiner operator for parallel operations.
Peek				
Synopsis	**Inspect the stream elements without disturbing the stream**			
Return	**Method**	**Cont**	**Type**	**Synopsis**
Stream<T>	peek(Consumer<? super T> action)	Intr	Inst	Performs the action defined in the consumer for each element in the stream without affecting the stream.

The IntStream Interface

Build				
Synopsis	**Create a stream**			
Variants	**Stream, LongStream, DoubleStream**			
Return	**Method**	**Cont**[4]	**Type**[5]	**Synopsis**
IntStream	concat(IntStream a, IntStream b)	Intr	Stat	Concatenates two streams to form a new one.
IntStream	empty()	Intr	Stat	Creates an empty stream.
IntStream	generate(IntSupplier s)	Intr	Stat	Creates a stream based on the given *IntSupplier*.
IntStream	iterate(final int seed, final IntUnaryOperator f)	Intr	Stat	Iterates through a stream from a starting point using *f* to generate the next element.
IntStream	of(int t)	Intr	Stat	Creates a stream of one int of value *t.*
IntStream	of(int... values)	Intr	Stat	Creates a stream of ints represented by *values.*
IntStream	range(int startInclusive, int endExclusive)	Intr	Stat	Creates a stream of ints from *startInclusive* to *endExclusive* exclusively.
IntStream	rangeClosed(int startInclusive, int endInclusive)	Intr	Stat	Creates a stream of ints from *startInclusive* to *endInclusive* inclusively.
DoubleStream	asDoubleStream()	Intr	Inst	Converts the stream into a stream of doubles.
LongStream	asLongStream()	Intr	Inst	Converts the stream into a stream of longs.
Stream<Integer>	boxed()	Intr	Inst	Converts the stream into a stream boxed into Integer.
S	onClose(Runnable closeHandler)	Intr	Inst	Returns an equivalent stream with an additional *closeHandler.*
IntStream	parallel()	Intr	Inst	Returns a parallel representation of the stream.
IntStream	sequential()	Intr	Inst	Returns a sequential representation of the stream.
IntStream	skip(long n)	Intr	Inst	Discards the first *n* elements and returns the remainder of the stream.
IntStream	sorted()	Intr	Inst	Sorts the stream based on its natural order.
Builder	builder()	Term	Stat	Returns a builder allowing the stream to be mutated.
void	close()	Term	Inst	Closes this stream, causing all close handlers for this stream pipeline to be called.
PrimitiveIterator. OfInt	iterator()	Term	Inst	Returns an *iterator.*

[4] Continuity: **Intr**: Intermediate, **Term**: Terminal
[5] Type: **Inst**: Instance, **Stat**: Static

Spliterator.OfInt	spliterator()	Term	Inst	Returns a *spliterator*
int[]	toArray()	Term	Inst	Converts the stream into an array of ints.
Iterate				
Synopsis	**Traverse a stream**			
Return	**Method**	**Cont**	**Type**	**Synopsis**
void	forEach(IntConsumer action)	Term	Inst	Iterates through the stream applying the *IntConsumer* to each element.
void	forEachOrdered(IntConsumer action)	Term	Inst	Iterates through the stream applying the *IntConsumer* function to each element and guaranteeing the order of the stream if a stream has one.
Filter				
Synopsis	**Filter stream elements**			
Return	**Method**	**Cont**	**Type**	**Synopsis**
IntStream	distinct()	Intr	Inst	Filters out duplicate elements and creates a new stream.
IntStream	filter(Predicate<? super T> predicate)	Intr	Inst	Filters elements of the stream based on the predicate condition.
IntStream	limit(long maxSize)	Intr	Inst	Returns a stream consisting of the elements of this stream, truncated to be no longer that maxSize.
Map				
Synopsis	**Transform elements of the stream**			
Return	**Method**	**Cont**	**Type**	**Synopsis**
IntStream	flatMap(IntFunction<? extends IntStream> mapper)	Intr	Inst	Returns an *IntStream* consisting of the results of replacing each element of this stream with the contents of the stream produced by applying the provided mapping function to each element.
IntStream	map(IntUnaryOperator mapper)	Intr	Inst	Returns an *IntStream* consisting of the results of applying the given function to the elements of this stream.
DoubleStream	mapToDouble(IntToDoubleFunction mapper)	Intr	Inst	Returns a *DoubleStream* consisting of the results of applying the given function to the elements of this stream.

LongStream	mapToLong(ToLongFunction<? super T> mapper)	Intr	Inst	Returns a *LongStream* consisting of the results of applying the given function to the elements of this stream.
Stream<U>	mapToObj(IntFunction<? extends U> mapper)	Intr	Inst	Returns an object-valued Stream consisting of the results of applying the given function to the elements of this stream.
Reduce				
Synopsis	**Reduce the stream to a value**			
Return	**Method**	**Cont**	**Type**	**Synopsis**
boolean	allMatch(IntPredicate predicate)	Term	Inst	Returns true if all elements in the stream match the *Predicate* condition.
boolean	anyMatch(IntPredicate predicate)	Term	Inst	Returns true if at least one elements in the stream matches the *Predicate* condition.
OptionalDouble	average()	Term	Inst	Returns the average value of the stream.
R	collect(Supplier<R> supplier, ObjIntConsumer<R> accumulator,BiConsumer<R, R> combiner)	Term	Inst	Aggregates the stream using a *Supplier*, *Accumulator,* and *Combiner.*
long	count()	Term	Inst	Counts the elements in the stream.
OptionalInt	findAny()	Term	Inst	Returns any element in the stream as an *Optional*. Does not necessarily return the first element.
OptionalInt	findFirst()	Term	Inst	Returns the first element of the stream as an *Optional*.
Optional<T>	max()	Term	Inst	Returns the maximum value in the stream.
Optional<T>	min()	Term	Inst	Returns the minimum value in the stream.
boolean	noneMatch(IntPredicate predicate)	Term	Inst	Returns true if no element in the stream matches the *Predicate* condition.
OptionalInt	reduce(IntBinaryOperator op)	Term	Inst	Aggregates the stream as one value.

int	reduce(int identity, IntBinaryOperator op)	Term	Inst	Aggregates the stream as one value using identity as the starting value.
int	sum()	Term	Inst	Adds all the elements of the stream.
IntSummaryStatistics	summaryStatistics()	Term	Inst	Returns an *IntSummaryStatistics* describing various summary data about the elements of this stream.
Peek				
Synopsis	**Inspect the stream elements without disturbing the stream**			
Return	**Method**	**Cont**	**Type**	**Synopsis**
IntStream	peek(IntConsumer action)	Intr	Inst	Performs the action defined in the consumer for each element in the stream without affecting the stream.

The functionalized Collections library

A list of methods added to key interfaces in the Collections library in Java 8.

Collection

Return	**Method**	**Synopsis**
boolean	removeIf(Predicate<? super E> filter)	Removes the element if the predicate condition is true.
Spliterator<E>	spliterator()	Creates a spliterator from the collection.
Stream<E>	stream()	Creates a stream from the collection.
Stream<E>	parallelStream()	Creates a parallel stream (if possible) from the collection.

List (extends Collection)

Return	**Method**	**Synopsis**
void	sort(Comparator<? super E> c)	Sorts the list using the comparator.
void	replaceAll(UnaryOperator<E> operator)	Replaces each element of this list with the result of applying the operator to that element.
Spliterator<E>	spliterator()	Creates a spliterator from the list.

Set (extends Collection)

Return	**Method**	**Synopsis**
Spliterator<E>	spliterator()	Creates a spliterator from the set.

Map

Return	**Method**	**Synopsis**
void	forEach(BiConsumer<? super K, ? super V> action)	Performs the given action on each entry in this map.
void	replaceAll(BiFunction<? super K, ? super V, ? extends V> function)	Replaces each entry's value with the result of invoking the given function on that entry.
V	getOrDefault(Object key, V defaultValue)	Returns the value to which the specified key is mapped or *defaultValue* if none mapped.
V	putIfAbsent(K key, V value)	Puts the element if absent.
V	computeIfAbsent(K key, Function<? super K, ? extends V> mappingFunction)	Puts the value generated by the mapping function if the key is absent.
V	computeIfPresent(K key, BiFunction<? super K, ? super V, ? extends V> remappingFunction)	If the value for the specified key is present and non-null, attempts to compute a new mapping given the key and its current mapped value.
V	compute(K key, BiFunction<? super K, ? super V, ? extends V> remappingFunction)	Attempts to compute a mapping for the specified key and its current mapped value or null if there is no current mapping.
V	merge(K key, V value, BiFunction<? super V, ? super V, ? extends V> remappingFunction)	If the specified key is not already associated with a value or is associated with null, associates it with the given value.
boolean	remove(Object key, Object value)	Removes the entry for the specified key only if it is currently mapped to the specified value.
boolean	replace(K key, V oldValue, V newValue)	Replaces the entry for the specified key only if it is currently mapped to the specified value.
V	replace(K key, V value)	Replaces the entry for the specified key only if it is currently mapped

		to some value.

Iterator		
Return	**Method**	**Synopsis**
void	forEachRemaining(Consumer<? super E> action)	Performs the given action for each remaining element in the order in which elements occur when iterating.

Iterable		
Return	**Method**	**Synopsis**
void	forEach(Consumer<? super T> action)	Performs the given action on the contents of the Iterable in the order in which elements occur when iterating.
Spliterator<T>	spliterator()	Creates a spliterator from the iterable.

The Optional Interface

Optional		
Synopsis	**Provide contingencies when returning values from methods.**	
Variants	**OptionDouble, OptionalInt, OptionalLong**	
Return	**Method**	**Synopsis**
T	get()	Returns the value contained in the *Optional* if present; otherwise throws a *NoSuchElementException*
void	ifPresent(Consumer<? super T> consumer)	Invokes the *Consumer* if present; otherwise does nothing
boolean	isPresent()	Returns true if a value is present in this *Optional*; otherwise returns false
T	orElse(T other)	Returns the value contained in this *Optional* if present; otherwise returns *other*
T	orElseGet(Supplier<? extends T> other)	Returns the value contained in this *Optional* if present; otherwise invokes other to generate a value
T	orElseThrow(Supplier<? extends X> exceptionSupplier) throws X	Returns the contained value if present; otherwise throws an exception to be created by the provided supplier
Optional<U>	map(Function<? super T, ? extends U> mapper)	Applies the provided *mapper* if a value is present. If the result is non-null, returns an *Optional*; otherwise returns an empty *Optional*.
Optional<U>	flatMap(Function<? super T, Optional<U>> mapper)	Applies the provided *mapper* if a value is present. If the result is non-null, returns an *Optional*; otherwise returns an empty *Optional*.
Optional<T>	filter(Predicate<? super T> predicate)	Returns an Optional if a value is present and matches the given *predicate*; otherwise returns an empty *Optional*
Optional<T>	empty()	Returns an empty *Optional* instance
Optional<T>	of(T value)	Returns an *Optional* with the specified present non-null value
Optional<U>	ofNullable(T value)	Returns an *Optional* describing the specified value if non-null; otherwise returns an empty *Optional*

The OptionalInt Interface

OptionalInt		
Synopsis	**Provide contingencies when returning values from methods.**	
Variants	**Optional, OptionalDouble, OptionalLong**	
Return	**Method**	**Synopsis**
int	get()	Returns the value contained in the *OptionalInt* if present; otherwise throws a *NoSuchElementException.*
void	ifPresent(IntConsumer consumer)	Invokes the *IntConsumer* if present; otherwise does nothing.
boolean	isPresent()	Returns true if a value is present in this *OptionalInt*; otherwise returns false.
int	orElse(int other)	Returns the value contained in this *OptionalInt* if present; otherwise returns *other.*
int	orElseGet(IntSupplier other)	Returns the value contained in this *OptionalInt* if present; otherwise invokes other to generate a value.
int	orElseThrow(Supplier<X> exceptionSupplier) throws X	Returns the contained value if present; otherwise throws an exception to be created by the provided supplier.
OptionalInt	empty()	Returns an empty *OptionalInt* instance.
OptionalInt	of(int value)	Returns an OptionalInt with the specified present non-null value.