Функциональные интерфейсы

- java.util.function
- @FunctionalInterface

Predicate<T>

- boolean test(T t)
- default Predicate<T> and(Predicate<? super T> other)
- default Predicate<T> negate()
- default Predicate<T> or(Predicate<? super T> other)
- static <T> Predicate<T> isEqual(Object targetRef)
- static <T> Predicate<T> not(Predicate<? super T> target)

IntPredicate

- boolean test(int value)
- default IntPredicate and(IntPredicate other)
- default IntPredicate negate()
- default IntPredicate or(IntPredicate other)

LongPredicate

- boolean test(long value)
- default LongPredicate and(LongPredicate other)
- default LongPredicate negate()
- default LongPredicate or(LongPredicate other)

DoublePredicate

- boolean test(double value)
- default DoublePredicate and(DoublePredicate other)
- default DoublePredicate negate()
- default DoublePredicate or(DoublePredicate other)

BiPredicate<T, U>

- boolean test(T t, U u)
- default BiPredicate<T, U> and(
  BiPredicate<? super T, ? super U> other)

- `default BiPredicate<T, U> negate()`
- `default BiPredicate<T, U> or(BiPredicate<? super T, ? super U> other)`

Consumer<T>

- `void accept(T t)`
- `default Consumer<T> andThen(Consumer<? super T> after)`

IntConsumer

- `void accept(int value)`
- `default IntConsumer andThen(IntConsumer after)`

LongConsumer

- `void accept(long value)`
- `default LongConsumer andThen(LongConsumer after)`

DoubleConsumer

- `void accept(double value)`
- `default DoubleConsumer andThen(DoubleConsumer after)`

BiConsumer<T, U>

- `void accept(T t, U u)`
- `default BiConsumer<T, U> andThen(BiConsumer<? Super T, ? super U> after)`

ObjIntConsumer<T>

- `void accept(T t, int value)`

ObjLongConsumer<T>

- `void accept(T t, long value)`

ObjDoubleConsumer<T>

- `void accept(T t, double value)`

Supplier<T>

- `T get()`

BooleanSupplier

- `boolean getAsBoolean()`

IntSupplier

- int getAsInt()

LongSupplier

- long getAsLong()

DoubleSupplier

- double getAsDouble()

Function<T, R>

- R apply(T t)
- default <V> Function<V, R> compose(
  Function<? super V, ? extends T> before)
- default <V> Function<T, V> andThen(
  Function<? super R, ? extends V> after)
- static <T> Function<T, T> identity()

IntFunction<R>

- R apply(int value)

LongFunction<R>

- R apply(long value)

DoubleFunction<R>

- R apply(double value)

ToIntFunction<T>

- int applyAsInt(T value)

ToLongFunction<T>

- long applyAsLong(T value)

ToDoubleFunction<T>

- double applyAsDouble(T value)

IntToLongFunction

- long applyAsLong(int value)

IntToDoubleFunction

- double applyAsDouble(int value)

LongToIntFunction

- `int applyAsInt(long value)`

LongToDoubleFunction

- `double applyAsDouble(long value)`

DoubleToIntFunction

- `int applyAsInt(double value)`

DoubleToLongFunction

- `long applyAsLong(double value)`

UnaryOperator<T>

- `extends Function<T, T>`
- `static <T> UnaryOperator<T> identity()`

IntUnaryOperator

- `int applyAsInt(int operand)`
- `default IntUnaryOperator compose( IntUnaryOperator before)`
- `default IntUnaryOperator andThen(IntUnaryOperator after)`
- `static IntUnaryOperator identity()`

LongUnaryOperator

- `long applyAsLong(long operand)`
- `default LongUnaryOperator compose( LongUnaryOperator before)`
- `default LongUnaryOperator andThen(LongUnaryOperator after)`
- `static LongUnaryOperator identity()`

DoubleUnaryOperator

- `double applyAsDouble(double operand)`
- `default DoubleUnaryOperator compose( DoubleUnaryOperator before)`
- `default DoubleUnaryOperator andThen( DoubleUnaryOperator after)`
- `static DoubleUnaryOperator identity()`

BiFunction<T, U, R>

- `R apply(T t, U u)`
- `default <V> BiFunction<T, U, V> andThen( Function<? super R, ? extends V> after)`

BinaryOperator<T>

- `extends BiFunction<T,T,T>`
- `public static <T> BinaryOperator <T>minBy( Comparator<? super T> comparator)`
- `public static <T> BinaryOperator<T> maxBy( Comparator<? super T> comparator)`

ToIntBiFunction<T,U>

- `int applyAsInt(T t, U u)`

ToLongBiFunction<T, U>

- `long applyAsLong(T t, U u)`

ToDoubleBiFunction<T, U>

- `double applyAsDouble(T t, U u)`

IntBinaryOperator

- `int applyAsInt(int left, int right)`

LongBinaryOperator

- `long applyAsLong(long left, long right)`

DoubleBinaryOperator

- `double applyAsDouble(double left, double right)`