Range.java

```
/* ===========================================================
 * JFreeChart : a free chart library for the Java(tm) platform
 * ===========================================================
 *
 * (C) Copyright 2000-2014, by Object Refinery Limited and Contributors.
 *
 * Project Info:  http://www.jfree.org/jfreechart/index.html
 *
 * This library is free software; you can redistribute it and/or modify it
 * under the terms of the GNU Lesser General Public License as published by
 * the Free Software Foundation; either version 2.1 of the License, or
 * (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
 * or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
 * License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301,
 * USA.
 *
 * [Oracle and Java are registered trademarks of Oracle and/or its affiliates.
 * Other names may be trademarks of their respective owners.]
 *
 * ----------
 * Range.java
 * ----------
 * (C) Copyright 2002-2014, by Object Refinery Limited and Contributors.
 *
 * Original Author:  David Gilbert (for Object Refinery Limited);
 * Contributor(s):   Chuanhao Chiu;
 *                   Bill Kelemen;
 *                   Nicolas Brodu;
 *                   Sergei Ivanov;
 *
 * Changes (from 23-Jun-2001)
 * --------------------------
 * 22-Apr-2002 : Version 1, loosely based by code by Bill Kelemen (DG);
 * 30-Apr-2002 : Added getLength() and getCentralValue() methods.  Changed
 *               argument check in constructor (DG);
 * 13-Jun-2002 : Added contains(double) method (DG);
 * 22-Aug-2002 : Added fix to combine method where both ranges are null, thanks
 *               to Chuanhao Chiu for reporting and fixing this (DG);
 * 07-Oct-2002 : Fixed errors reported by Checkstyle (DG);
 * 26-Mar-2003 : Implemented Serializable (DG);
 * 14-Aug-2003 : Added equals() method (DG);
 * 27-Aug-2003 : Added toString() method (BK);
 * 11-Sep-2003 : Added Clone Support (NB);
 * 23-Sep-2003 : Fixed Checkstyle issues (DG);
 * 25-Sep-2003 : Oops, Range immutable, clone not necessary (NB);
 * 05-May-2004 : Added constrain() and intersects() methods (DG);
 * 18-May-2004 : Added expand() method (DG);
 * ------------- JFreeChart 1.0.x ---------------------------------------------
 * 11-Jan-2006 : Added new method expandToInclude(Range, double) (DG);
 * 18-Dec-2007 : New methods intersects(Range) and scale(...) thanks to Sergei
 *               Ivanov (DG);
 * 08-Jan-2012 : New method combineIgnoringNaN() (DG);
 * 23-Feb-2014 : Added isNaNRange() method (DG);
 *
 */

package org.jfree.data;

import java.io.Serializable;
import org.jfree.chart.util.ParamChecks;

/**
 * Represents an immutable range of values.
 */
public strictfp class Range implements Serializable {

    /** For serialization. */
    private static final long serialVersionUID = -906333695431863380L;

    /** The lower bound of the range. */
    private double lower;

    /** The upper bound of the range. */
    private double upper;

    /**
     * Creates a new range.
     *
     * @param lower  the lower bound (must be &lt;= upper bound).
     * @param upper  the upper bound (must be &gt;= lower bound).
     */
    public Range(double lower, double upper) {
2       if (lower > upper) {
            String msg = "Range(double, double): require lower (" + lower
                + ") <= upper (" + upper + ").";
            throw new IllegalArgumentException(msg);
        }
        this.lower = lower;
        this.upper = upper;
    }

    /**
     * Returns the lower bound for the range.
     *
```

```
       * @return The lower bound.
       */
      public double getLowerBound() {
1         return this.lower;
      }

      /**
       * Returns the upper bound for the range.
       *
       * @return The upper bound.
       */
      public double getUpperBound() {
1         return this.upper;
      }

      /**
       * Returns the length of the range.
       *
       * @return The length.
       */
      public double getLength() {
2         return this.upper - this.lower;
      }

      /**
       * Returns the central value for the range.
       *
       * @return The central value.
       */
      public double getCentralValue() {
4         return this.lower / 2.0 + this.upper / 2.0;
      }

      /**
       * Returns <code>true</code> if the range contains the specified value and
       * <code>false</code> otherwise.
       *
       * @param value  the value to lookup.
       *
       * @return <code>true</code> if the range contains the specified value.
       */
      public boolean contains(double value) {
6         return (value >= this.lower && value <= this.upper);
      }

      /**
       * Returns <code>true</code> if the range intersects with the specified
       * range, and <code>false</code> otherwise.
       *
       * @param b0  the lower bound (should be &lt;= b1).
       * @param b1  the upper bound (should be &gt;= b0).
       *
       * @return A boolean.
       */
      public boolean intersects(double b0, double b1) {
2         if (b0 <= this.lower) {
4             return (b1 > this.lower);
          }
          else {
6             return (b0 < this.upper && b1 >= b0);
          }
      }

      /**
       * Returns <code>true</code> if the range intersects with the specified
       * range, and <code>false</code> otherwise.
       *
       * @param range  another range (<code>null</code> not permitted).
       *
       * @return A boolean.
       *
       * @since 1.0.9
       */
      public boolean intersects(Range range) {
2         return intersects(range.getLowerBound(), range.getUpperBound());
      }

      /**
       * Returns the value within the range that is closest to the specified
       * value.
       *
       * @param value  the value.
       *
       * @return The constrained value.
       */
      public double constrain(double value) {
          double result = value;
1         if (!contains(value)) {
2             if (value > this.upper) {
                  result = this.upper;
              }
2             else if (value < this.lower) {
                  result = this.lower;
              }
          }
1         return result;
      }

      /**
       * Creates a new range by combining two existing ranges.
       * <P>
       * Note that:
       * <ul>
       *   <li>either range can be <code>null</code>, in which case the other
       *       range is returned;</li>
       *   <li>if both ranges are <code>null</code> the return value is
       *       <code>null</code>.</li>
       * </ul>
       *
       * @param range1  the first range (<code>null</code> permitted).
       * @param range2  the second range (<code>null</code> permitted).
       *
       * @return A new range (possibly <code>null</code>).
       */
      public static Range combine(Range range1, Range range2) {
1         if (range1 == null) {
1             return range2;
          }
1         if (range2 == null) {
1             return range1;
          }
          double l = Math.min(range1.getLowerBound(), range2.getLowerBound());
          double u = Math.max(range1.getUpperBound(), range2.getUpperBound());
1         return new Range(l, u);
      }

      /**
       * Returns a new range that spans both <code>range1</code> and
       * <code>range2</code>.  This method has a special handling to ignore
       * Double.NaN values.
       *
       * @param range1  the first range (<code>null</code> permitted).
       * @param range2  the second range (<code>null</code> permitted).
       *
       * @return A new range (possibly <code>null</code>).
       *
       * @since 1.0.15
       */
      public static Range combineIgnoringNaN(Range range1, Range range2) {
1         if (range1 == null) {
2             if (range2 != null && range2.isNaNRange()) {
                  return null;
              }
1             return range2;
          }
1         if (range2 == null) {
1             if (range1.isNaNRange()) {
                  return null;
              }
1             return range1;
          }
          double l = min(range1.getLowerBound(), range2.getLowerBound());
          double u = max(range1.getUpperBound(), range2.getUpperBound());
2         if (Double.isNaN(l) && Double.isNaN(u)) {
              return null;
          }
1         return new Range(l, u);
      }

      /**
       * Returns the minimum value.  If either value is NaN, the other value is
       * returned.  If both are NaN, NaN is returned.
       *
       * @param d1  value 1.
       * @param d2  value 2.
       *
       * @return The minimum of the two values.
       */
      private static double min(double d1, double d2) {
1         if (Double.isNaN(d1)) {
1             return d2;
          }
1         if (Double.isNaN(d2)) {
1             return d1;
          }
1         return Math.min(d1, d2);
      }

      private static double max(double d1, double d2) {
1         if (Double.isNaN(d1)) {
1             return d2;
          }
1         if (Double.isNaN(d2)) {
1             return d1;
          }
1         return Math.max(d1, d2);
      }

      /**
       * Returns a range that includes all the values in the specified
       * <code>range</code> AND the specified <code>value</code>.
       *
       * @param range  the range (<code>null</code> permitted).
       * @param value  the value that must be included.
       *
       * @return A range.
       *
       * @since 1.0.1
       */
      public static Range expandToInclude(Range range, double value) {
1         if (range == null) {
1             return new Range(value, value);
          }
2         if (value < range.getLowerBound()) {
1             return new Range(value, range.getUpperBound());
          }
2         else if (value > range.getUpperBound()) {
1             return new Range(range.getLowerBound(), value);
          }
          else {
1             return range;
          }
      }

      /**
       * Creates a new range by adding margins to an existing range.
       *
       * @param range  the range (<code>null</code> not permitted).
       * @param lowerMargin  the lower margin (expressed as a percentage of the
       *                     range length).
       * @param upperMargin  the upper margin (expressed as a percentage of the
       *                     range length).
       *
       * @return The expanded range.
       */
      public static Range expand(Range range,
                                 double lowerMargin, double upperMargin) {
1         ParamChecks.nullNotPermitted(range, "range");
          double length = range.getLength();
2         double lower = range.getLowerBound() - length * lowerMargin;
2         double upper = range.getUpperBound() + length * upperMargin;
2         if (lower > upper) {
3             lower = lower / 2.0 + upper / 2.0;
              upper = lower;
          }
1         return new Range(lower, upper);
      }

      /**
       * Shifts the range by the specified amount.
       *
       * @param base  the base range (<code>null</code> not permitted).
       * @param delta  the shift amount.
       *
       * @return A new range.
       */
      public static Range shift(Range base, double delta) {
1         return shift(base, delta, false);
      }

      /**
       * Shifts the range by the specified amount.
       *
       * @param base  the base range (<code>null</code> not permitted).
       * @param delta  the shift amount.
       * @param allowZeroCrossing  a flag that determines whether or not the
       *                           bounds of the range are allowed to cross
       *                           zero after adjustment.
       *
       * @return A new range.
       */
      public static Range shift(Range base, double delta,
                                boolean allowZeroCrossing) {
1         ParamChecks.nullNotPermitted(base, "base");
1         if (allowZeroCrossing) {
```

```
2               return new Range(base.getLowerBound() + delta,
1                       base.getUpperBound() + delta);
            }
            else {
1               return new Range(shiftWithNoZeroCrossing(base.getLowerBound(),
                        delta), shiftWithNoZeroCrossing(base.getUpperBound(),
                        delta));
            }
        }

        /**
         * Returns the given <code>value</code> adjusted by <code>delta</code> but
         * with a check to prevent the result from crossing <code>0.0</code>.
         *
         * @param value  the value.
         * @param delta  the adjustment.
         *
         * @return The adjusted value.
         */
        private static double shiftWithNoZeroCrossing(double value, double delta) {
2           if (value > 0.0) {
2               return Math.max(value + delta, 0.0);
            }
2           else if (value < 0.0) {
2               return Math.min(value + delta, 0.0);
            }
            else {
2               return value + delta;
            }
        }

        /**
         * Scales the range by the specified factor.
         *
         * @param base the base range (<code>null</code> not permitted).
         * @param factor the scaling factor (must be non-negative).
         *
         * @return A new range.
         *
         * @since 1.0.9
         */
        public static Range scale(Range base, double factor) {
1           ParamChecks.nullNotPermitted(base, "base");
2           if (factor < 0) {
                throw new IllegalArgumentException("Negative 'factor' argument.");
            }
2           return new Range(base.getLowerBound() * factor,
1                   base.getUpperBound() * factor);
        }

        /**
         * Tests this object for equality with an arbitrary object.
         *
```

```
420        * @param obj  the object to test against (<code>null</code> permitted).
421        *
422        * @return A boolean.
423        */
424       @Override
425       public boolean equals(Object obj) {
426 1         if (!(obj instanceof Range)) {
427 1             return false;
428           }
429           Range range = (Range) obj;
430 1         if (!(this.lower == range.lower)) {
431 1             return false;
432           }
433 1         if (!(this.upper == range.upper)) {
434 1             return false;
435           }
436 1         return true;
437       }
438
439       /**
440        * Returns <code>true</code> if both the lower and upper bounds are
441        * <code>Double.NaN</code>, and <code>false</code> otherwise.
442        *
443        * @return A boolean.
444        *
445        * @since 1.0.18
446        */
447       public boolean isNaNRange() {
448 4         return Double.isNaN(this.lower) && Double.isNaN(this.upper);
449       }
450
451       /**
452        * Returns a hash code.
453        *
454        * @return A hash code.
455        */
456       @Override
457       public int hashCode() {
458           int result;
459           long temp;
460           temp = Double.doubleToLongBits(this.lower);
461 2         result = (int) (temp ^ (temp >>> 32));
462           temp = Double.doubleToLongBits(this.upper);
463 4         result = 29 * result + (int) (temp ^ (temp >>> 32));
464 1         return result;
465       }
466
467       /**
468        * Returns a string representation of this Range.
469        *
470        * @return A String "Range[lower,upper]" where lower=lower range and
471        *         upper=upper range.
472        */
```

```
473         @Override
474         public String toString() {
475 1           return ("Range[" + this.lower + "," + this.upper + "]");
476         }
477
478     }
```

Mutations

90	1. changed conditional boundary → KILLED 2. negated conditional → KILLED
105	1. replaced double return with 0.0d for org/jfree/data/Range::getLowerBound → KILLED
114	1. replaced double return with 0.0d for org/jfree/data/Range::getUpperBound → KILLED
123	1. Replaced double subtraction with addition → KILLED 2. replaced double return with 0.0d for org/jfree/data/Range::getLength → KILLED
132	1. Replaced double division with multiplication → KILLED 2. Replaced double division with multiplication → KILLED 3. Replaced double addition with subtraction → KILLED 4. replaced double return with 0.0d for org/jfree/data/Range::getCentralValue → KILLED
144	1. replaced boolean return with false for org/jfree/data/Range::contains → KILLED 2. replaced boolean return with true for org/jfree/data/Range::contains → KILLED 3. changed conditional boundary → KILLED 4. changed conditional boundary → KILLED 5. negated conditional → KILLED 6. negated conditional → KILLED
157	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED
158	1. replaced boolean return with false for org/jfree/data/Range::intersects → KILLED 2. replaced boolean return with true for org/jfree/data/Range::intersects → KILLED 3. changed conditional boundary → SURVIVED 4. negated conditional → KILLED
161	1. replaced boolean return with false for org/jfree/data/Range::intersects → KILLED 2. replaced boolean return with true for org/jfree/data/Range::intersects → KILLED 3. changed conditional boundary → SURVIVED 4. changed conditional boundary → SURVIVED 5. negated conditional → KILLED 6. negated conditional → KILLED
176	1. replaced boolean return with false for org/jfree/data/Range::intersects → KILLED 2. replaced boolean return with true for org/jfree/data/Range::intersects → SURVIVED
189	1. negated conditional → KILLED
190	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED
193	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED
197	1. replaced double return with 0.0d for org/jfree/data/Range::constrain → KILLED
217	1. negated conditional → KILLED

```
218 1. replaced return value with null for org/jfree/data/Range::combine → KILLED
220 1. negated conditional → KILLED
221 1. replaced return value with null for org/jfree/data/Range::combine → KILLED
225 1. replaced return value with null for org/jfree/data/Range::combine → KILLED
241 1. negated conditional → KILLED
242 1. negated conditional → KILLED
    2. negated conditional → KILLED
245 1. replaced return value with null for org/jfree/data/Range::combineIgnoringNaN
    → KILLED
247 1. negated conditional → KILLED
248 1. negated conditional → KILLED
251 1. replaced return value with null for org/jfree/data/Range::combineIgnoringNaN
    → KILLED
255 1. negated conditional → KILLED
    2. negated conditional → KILLED
258 1. replaced return value with null for org/jfree/data/Range::combineIgnoringNaN
    → KILLED
271 1. negated conditional → KILLED
272 1. replaced double return with 0.0d for org/jfree/data/Range::min → KILLED
274 1. negated conditional → KILLED
275 1. replaced double return with 0.0d for org/jfree/data/Range::min → KILLED
277 1. replaced double return with 0.0d for org/jfree/data/Range::min → KILLED
281 1. negated conditional → KILLED
282 1. replaced double return with 0.0d for org/jfree/data/Range::max → KILLED
284 1. negated conditional → KILLED
285 1. replaced double return with 0.0d for org/jfree/data/Range::max → KILLED
287 1. replaced double return with 0.0d for org/jfree/data/Range::max → KILLED
302 1. negated conditional → KILLED
303 1. replaced return value with null for org/jfree/data/Range::expandToInclude →
    KILLED
305 1. changed conditional boundary → SURVIVED
    2. negated conditional → KILLED
306 1. replaced return value with null for org/jfree/data/Range::expandToInclude →
    KILLED
308 1. changed conditional boundary → SURVIVED
    2. negated conditional → KILLED
309 1. replaced return value with null for org/jfree/data/Range::expandToInclude →
    KILLED
312 1. replaced return value with null for org/jfree/data/Range::expandToInclude →
    KILLED
329 1. removed call to org/jfree/chart/util/ParamChecks::nullNotPermitted →
    SURVIVED
331 1. Replaced double multiplication with division → KILLED
    2. Replaced double subtraction with addition → KILLED
332 1. Replaced double multiplication with division → KILLED
    2. Replaced double addition with subtraction → KILLED
333 1. changed conditional boundary → SURVIVED
    2. negated conditional → KILLED
    1. Replaced double division with multiplication → KILLED
334 2. Replaced double division with multiplication → KILLED
    3. Replaced double addition with subtraction → KILLED
337 1. replaced return value with null for org/jfree/data/Range::expand → KILLED
349 1. replaced return value with null for org/jfree/data/Range::shift → KILLED
365 1. removed call to org/jfree/chart/util/ParamChecks::nullNotPermitted →
    SURVIVED
366 1. negated conditional → KILLED
```

```
367  1. Replaced double addition with subtraction → KILLED
     2. replaced return value with null for org/jfree/data/Range::shift → KILLED
368  1. Replaced double addition with subtraction → KILLED
371  1. replaced return value with null for org/jfree/data/Range::shift → KILLED
387  1. changed conditional boundary → KILLED
     2. negated conditional → KILLED
388  1. Replaced double addition with subtraction → KILLED
     2. replaced double return with 0.0d for
     org/jfree/data/Range::shiftWithNoZeroCrossing → KILLED
390  1. changed conditional boundary → SURVIVED
     2. negated conditional → KILLED
391  1. Replaced double addition with subtraction → KILLED
     2. replaced double return with 0.0d for
     org/jfree/data/Range::shiftWithNoZeroCrossing → KILLED
394  1. Replaced double addition with subtraction → KILLED
     2. replaced double return with 0.0d for
     org/jfree/data/Range::shiftWithNoZeroCrossing → KILLED
409  1. removed call to org/jfree/chart/util/ParamChecks::nullNotPermitted →
     SURVIVED
410  1. changed conditional boundary → SURVIVED
     2. negated conditional → KILLED
413  1. Replaced double multiplication with division → KILLED
     2. replaced return value with null for org/jfree/data/Range::scale → KILLED
414  1. Replaced double multiplication with division → KILLED
426  1. negated conditional → KILLED
427  1. replaced boolean return with true for org/jfree/data/Range::equals → KILLED
430  1. negated conditional → KILLED
431  1. replaced boolean return with true for org/jfree/data/Range::equals → KILLED
433  1. negated conditional → KILLED
434  1. replaced boolean return with true for org/jfree/data/Range::equals → KILLED
436  1. replaced boolean return with false for org/jfree/data/Range::equals → KILLED
448  1. replaced boolean return with false for org/jfree/data/Range::isNaNRange →
     KILLED
     2. replaced boolean return with true for org/jfree/data/Range::isNaNRange →
     KILLED
     3. negated conditional → KILLED
     4. negated conditional → KILLED
461  1. Replaced Unsigned Shift Right with Shift Left → NO_COVERAGE
     2. Replaced XOR with AND → NO_COVERAGE
463  1. Replaced integer multiplication with division → NO_COVERAGE
     2. Replaced Unsigned Shift Right with Shift Left → NO_COVERAGE
     3. Replaced XOR with AND → NO_COVERAGE
     4. Replaced integer addition with subtraction → NO_COVERAGE
464  1. replaced int return with 0 for org/jfree/data/Range::hashCode → NO_COVERAGE
475  1. replaced return value with "" for org/jfree/data/Range::toString → KILLED
```

Active mutators

- BOOLEAN_FALSE_RETURN
- BOOLEAN_TRUE_RETURN
- CONDITIONALS_BOUNDARY_MUTATOR
- EMPTY_RETURN_VALUES
- INCREMENTS_MUTATOR
- INVERT_NEGS_MUTATOR
- MATH_MUTATOR
- NEGATE_CONDITIONALS_MUTATOR
- NULL_RETURN_VALUES
- PRIMITIVE_RETURN_VALS_MUTATOR
- VOID_METHOD_CALL_MUTATOR

Tests examined

- org.jfree.data.RangeTest_v3.constrainTestOutsideRangeAbove(org.jfree.data.RangeTest_v3) (4 ms)
- org.jfree.data.testA3.RangeTest_v2.expandToIncludeNull(org.jfree.data.testA3.RangeTest_v2) (2 ms)
- org.jfree.data.testA3.RangeTest_v2.expandEqual(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.RangeTest_v3.containsTestMax(org.jfree.data.RangeTest_v3) (0 ms)
- org.jfree.data.RangeTest_v3.shiftWithNoZeroCrossingWithValuesBelowZero(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.RangeTest_v3.ignoringnanFirstNullSecondNaN(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.combineTestNoOverlap(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.RangeTest_v3.intersectsInIn(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.intersectsOutOutLowHigh(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.RangeTest_v3.equalsFalseForNonRange(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.RangeTest_v3.combineTestInput1IsNull(org.jfree.data.RangeTest_v3) (0 ms)
- org.jfree.data.RangeTest_v3.expandEqual(org.jfree.data.RangeTest_v3) (3 ms)
- org.jfree.data.testA3.RangeTest_v2.centralValueShouldBeZero(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.RangeTest_v3.equalsTestForHigherRange(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.constrainTestOutsideRangeAbove(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.testA2.RangeTest.centralValueShouldBeNegative(org.jfree.data.testA2.RangeTest) (0 ms)
- org.jfree.data.testA2.RangeTest.constrainTestOnMax(org.jfree.data.testA2.RangeTest) (0 ms)
- org.jfree.data.testA2.RangeTest.constrainTestOutsideRangeBelow(org.jfree.data.testA2.RangeTest) (0 ms)
- org.jfree.data.testA3.RangeTest_v2.ignoringnanSecondNull(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.expandLowerBecomesBigger(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.testA2.RangeTest.getUpperBoundTest(org.jfree.data.testA2.RangeTest) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.centralValueShouldBePositive(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.RangeTest_v3.intersectsInOut(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.combineTestInput1IsNull(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.testA2.RangeTest.combineTestNoOverlap(org.jfree.data.testA2.RangeTest) (0 ms)
- org.jfree.data.RangeTest_v3.equalsTestForLowerRange(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.expandToIncludeInside(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.RangeTest_v3.intersectsOutOutLowHigh(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.RangeTest_v3.getLengthSameValues(org.jfree.data.RangeTest_v3) (4 ms)
- org.jfree.data.testA3.RangeTest_v2.ignoringnanLowerNan(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.RangeTest_v3.ignoringnanSecondNull(org.jfree.data.RangeTest_v3) (6 ms)
- org.jfree.data.RangeTest_v3.centralValueShouldBeZero(org.jfree.data.RangeTest_v3) (0 ms)
- org.jfree.data.testA2.RangeTest.containsTestMax(org.jfree.data.testA2.RangeTest) (0 ms)
- org.jfree.data.RangeTest_v3.switchedInputToConstructor(org.jfree.data.RangeTest_v3) (7 ms)
- org.jfree.data.RangeTest_v3.constrainTestOnMin(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.ignoringnanBothNull(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.testA3.RangeTest_v2.constrainTestOnUpper(org.jfree.data.testA3.RangeTest_v2) (6 ms)
- org.jfree.data.testA3.RangeTest_v2.combineTestIntersect(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.RangeTest_v3.shiftBasicValue(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.equalsTestForHigherRange(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.RangeTest_v3.combineWithOneNanUpper(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.equalsTestForLowerRange(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.RangeTest_v3.constrainTestOnUpper(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.RangeTest_v3.intersectsRange(org.jfree.data.RangeTest_v3) (0 ms)
- org.jfree.data.testA3.RangeTest_v2.scaleNegativeFactor(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.testA2.RangeTest.containsTestForOnLowerBound(org.jfree.data.testA2.RangeTest) (0 ms)
- org.jfree.data.testA3.RangeTest_v2.constrainTestOnLower(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.testA2.RangeTest.getLowerBoundTest(org.jfree.data.testA2.RangeTest) (0 ms)
- org.jfree.data.testA3.RangeTest_v2.containsTestForLessThanLowerBound(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.RangeTest_v3.expandToIncludeBelow(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.RangeTest_v3.toStringTest(org.jfree.data.RangeTest_v3) (4 ms)
- org.jfree.data.RangeTest_v3.scaleNegativeFactor(org.jfree.data.RangeTest_v3) (9 ms)
- org.jfree.data.testA2.RangeTest.combineTestIntersect(org.jfree.data.testA2.RangeTest) (0 ms)
- org.jfree.data.testA3.RangeTest_v2.constrainTestOnMin(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.RangeTest_v3.ignoringnanFirstNaNSecondNull(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.shiftWithZeroCrossing(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.RangeTest_v3.shiftWithNoZeroCrossingWithValuesAboveZero(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.testA2.RangeTest.containsTestForMoreThanUpperBound(org.jfree.data.testA2.RangeTest) (2 ms)

- org.jfree.data.testA3.RangeTest_v2.ignoringnanFirstNullSecondNaN(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.RangeTest_v3.combineTestInput2IsNull(org.jfree.data.RangeTest_v3) (2 ms)
- org.jfree.data.testA2.RangeTest.constrainTestOnUpper(org.jfree.data.testA2.RangeTest) (0 ms)
- org.jfree.data.testA2.RangeTest.equalsTestForSameRange(org.jfree.data.testA2.RangeTest) (0 ms)
- org.jfree.data.testA3.RangeTest_v2.shiftWithNoZeroCrossingWithValuesBelowZero (org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.RangeTest_v3.expandToIncludeInside(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.RangeTest_v3.ignoringnanFirstNull(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.intersectsInOut(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.RangeTest_v3.ignoringnanLowerNan(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.getLengthDifferentValues(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.testA3.RangeTest_v2.CombineIgnoringNaNBothNULL(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.scalePositiveFactor(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.testA3.RangeTest_v2.shiftWithNoZeroCrossingWithZeroValues (org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.testA2.RangeTest.combineTestNull(org.jfree.data.testA2.RangeTest) (1 ms)
- org.jfree.data.RangeTest_v3.containsTestForLessThanLowerBound(org.jfree.data.RangeTest_v3) (0 ms)
- org.jfree.data.testA2.RangeTest.containsTestMin(org.jfree.data.testA2.RangeTest) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.centralValueShouldBeNegative(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.testA2.RangeTest.equalsTestForLowerRange(org.jfree.data.testA2.RangeTest) (0 ms)
- org.jfree.data.testA2.RangeTest.centralValueShouldBePositive(org.jfree.data.testA2.RangeTest) (1 ms)
- org.jfree.data.RangeTest_v3.containsTestForOnLowerBound(org.jfree.data.RangeTest_v3) (0 ms)
- org.jfree.data.RangeTest_v3.getUpperBoundTest(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.constrainTestMiddleOfRange(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.RangeTest_v3.ignoringnanBothNull(org.jfree.data.RangeTest_v3) (0 ms)
- org.jfree.data.RangeTest_v3.centralValueShouldBePositive(org.jfree.data.RangeTest_v3) (0 ms)
- org.jfree.data.testA3.RangeTest_v2.containsTestForInBetweenBounds(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.expandToIncludeAbove(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.testA2.RangeTest.constrainTestMiddleOfRange(org.jfree.data.testA2.RangeTest) (0 ms)
- org.jfree.data.testA3.RangeTest_v2.constrainTestOnMax(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.intersectsRange(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.testA2.RangeTest.constrainTestOnLower(org.jfree.data.testA2.RangeTest) (0 ms)
- org.jfree.data.RangeTest_v3.equalsTestForSameRange(org.jfree.data.RangeTest_v3) (4 ms)
- org.jfree.data.RangeTest_v3.centralValueShouldBeNegative(org.jfree.data.RangeTest_v3) (0 ms)
- org.jfree.data.testA3.RangeTest_v2.ignoringnanIntersecting(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.testA3.RangeTest_v2.switchedInputToConstructor(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.testA3.RangeTest_v2.intersectsOutOutHigh(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.testA2.RangeTest.containsTestForOnUpperBound(org.jfree.data.testA2.RangeTest) (1 ms)
- org.jfree.data.RangeTest_v3.ignoringnanIntersecting(org.jfree.data.RangeTest_v3) (0 ms)
- org.jfree.data.RangeTest_v3.shiftWithZeroCrossing(org.jfree.data.RangeTest_v3) (0 ms)
- org.jfree.data.testA3.RangeTest_v2.combineTestInput2IsNull(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.shiftWithNoZeroCrossingWithValuesAboveZero (org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.equalsFalseForNonRange(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.RangeTest_v3.constrainTestMiddleOfRange(org.jfree.data.RangeTest_v3) (5 ms)
- org.jfree.data.testA3.RangeTest_v2.expandToIncludeBelow(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.RangeTest_v3.containsTestForInBetweenBounds(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.ignoringnanFirstNaNSecondNull(org.jfree.data.testA3.RangeTest_v2) (4 ms)
- org.jfree.data.testA3.RangeTest_v2.getLengthSameValues(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.RangeTest_v3.expandToIncludeAbove(org.jfree.data.RangeTest_v3) (3 ms)
- org.jfree.data.testA3.RangeTest_v2.intersectsOutIn(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.testA2.RangeTest.centralValueShouldBeZero(org.jfree.data.testA2.RangeTest) (0 ms)
- org.jfree.data.RangeTest_v3.containsTestMin(org.jfree.data.RangeTest_v3) (0 ms)
- org.jfree.data.testA3.RangeTest_v2.containsTestForOnUpperBound(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.testA3.RangeTest_v2.combineWithOneNanLower(org.jfree.data.testA3.RangeTest_v2) (2 ms)
- org.jfree.data.RangeTest_v3.combineTestNoOverlap(org.jfree.data.RangeTest_v3) (0 ms)
- org.jfree.data.RangeTest_v3.containsTestForMoreThanUpperBound(org.jfree.data.RangeTest_v3) (0 ms)
- org.jfree.data.testA3.RangeTest_v2.getUpperBoundTest(org.jfree.data.testA3.RangeTest_v2) (3 ms)
- org.jfree.data.testA2.RangeTest.equalsTestForHigherRange(org.jfree.data.testA2.RangeTest) (0 ms)
- org.jfree.data.RangeTest_v3.scalePositiveFactor(org.jfree.data.RangeTest_v3) (3 ms)
- org.jfree.data.RangeTest_v3.containsTestForOnUpperBound(org.jfree.data.RangeTest_v3) (0 ms)
- org.jfree.data.testA2.RangeTest.containsTestForLessThanLowerBound(org.jfree.data.testA2.RangeTest) (0 ms)

- org.jfree.data.RangeTest_v3.intersectsOutIn(org.jfree.data.RangeTest_v3) (6 ms)
- org.jfree.data.testA3.RangeTest_v2.containsTestMin(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.RangeTest_v3.getLengthDifferentValues(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.RangeTest_v3.ignoringnanBothNaN(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.containsTestMax(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.RangeTest_v3.expandToIncludeNull(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.RangeTest_v3.intersectsOutOutLow(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.toStringTest(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.RangeTest_v3.intersectsOutOutHigh(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.getLowerBoundTest(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.ignoringnanFirstNull(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.shiftBasicValue(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.ignoringnanBothNaN(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.RangeTest_v3.shiftWithNoZeroCrossingWithZeroValues(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.containsTestForOnLowerBound(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.RangeTest_v3.getLowerBoundTest(org.jfree.data.RangeTest_v3) (5 ms)
- org.jfree.data.testA3.RangeTest_v2.combineWithOneNanUpper(org.jfree.data.testA3.RangeTest_v2) (3 ms)
- org.jfree.data.testA2.RangeTest.constrainTestOutsideRangeAbove(org.jfree.data.testA2.RangeTest) (0 ms)
- org.jfree.data.RangeTest_v3.CombineIgnoringNaNBothNULL(org.jfree.data.RangeTest_v3) (4 ms)
- org.jfree.data.testA3.RangeTest_v2.equalsTestForSameRange(org.jfree.data.testA3.RangeTest_v2) (1 ms)
- org.jfree.data.RangeTest_v3.combineTestIntersect(org.jfree.data.RangeTest_v3) (6 ms)
- org.jfree.data.testA3.RangeTest_v2.intersectsOutOutLow(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.testA3.RangeTest_v2.containsTestForMoreThanUpperBound (org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.RangeTest_v3.constrainTestOutsideRangeBelow(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.RangeTest_v3.constrainTestOnLower(org.jfree.data.RangeTest_v3) (1 ms)
- org.jfree.data.testA3.RangeTest_v2.constrainTestOutsideRangeBelow(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.RangeTest_v3.combineWithOneNanLower(org.jfree.data.RangeTest_v3) (2 ms)
- org.jfree.data.testA2.RangeTest.containsTestForInBetweenBounds(org.jfree.data.testA2.RangeTest) (0 ms)
- org.jfree.data.testA3.RangeTest_v2.intersectsInIn(org.jfree.data.testA3.RangeTest_v2) (0 ms)
- org.jfree.data.testA2.RangeTest.constrainTestOnMin(org.jfree.data.testA2.RangeTest) (0 ms)
- org.jfree.data.RangeTest_v3.constrainTestOnMax(org.jfree.data.RangeTest_v3) (5 ms)
- org.jfree.data.RangeTest_v3.expandLowerBecomesBigger(org.jfree.data.RangeTest_v3) (1 ms)

Report generated by PIT 1.4.11