Range.java

```
1     /* ===========================================================
2      * JFreeChart : a free chart library for the Java(tm) platform
3      * ===========================================================
4      *
5      * (C) Copyright 2000-2014, by Object Refinery Limited and Contributors.
6      *
7      * Project Info:  http://www.jfree.org/jfreechart/index.html
8      *
9      * This library is free software; you can redistribute it and/or modify it
10     * under the terms of the GNU Lesser General Public License as published by
11     * the Free Software Foundation; either version 2.1 of the License, or
12     * (at your option) any later version.
13     *
14     * This library is distributed in the hope that it will be useful, but
15     * WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
16     * or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
17     * License for more details.
18     *
19     * You should have received a copy of the GNU Lesser General Public
20     * License along with this library; if not, write to the Free Software
21     * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301,
22     * USA.
23     *
24     * [Oracle and Java are registered trademarks of Oracle and/or its affiliates.
25     * Other names may be trademarks of their respective owners.]
26     *
27     * ----------
28     * Range.java
29     * ----------
30     * (C) Copyright 2002-2014, by Object Refinery Limited and Contributors.
31     *
32     * Original Author:  David Gilbert (for Object Refinery Limited);
33     * Contributor(s):   Chuanhao Chiu;
34     *                   Bill Kelemen;
35     *                   Nicolas Brodu;
36     *                   Sergei Ivanov;
37     *
38     * Changes (from 23-Jun-2001)
39     * --------------------------
40     * 22-Apr-2002 : Version 1, loosely based by code by Bill Kelemen (DG);
41     * 30-Apr-2002 : Added getLength() and getCentralValue() methods.  Changed
42     *               argument check in constructor (DG);
43     * 13-Jun-2002 : Added contains(double) method (DG);
44     * 22-Aug-2002 : Added fix to combine method where both ranges are null, thanks
45     *               to Chuanhao Chiu for reporting and fixing this (DG);
46     * 07-Oct-2002 : Fixed errors reported by Checkstyle (DG);
47     * 26-Mar-2003 : Implemented Serializable (DG);
48     * 14-Aug-2003 : Added equals() method (DG);
49     * 27-Aug-2003 : Added toString() method (BK);
50     * 11-Sep-2003 : Added Clone Support (NB);
51     * 23-Sep-2003 : Fixed Checkstyle issues (DG);
52     * 25-Sep-2003 : Oops, Range immutable, clone not necessary (NB);
53     * 05-May-2004 : Added constrain() and intersects() methods (DG);
54     * 18-May-2004 : Added expand() method (DG);
```

```
55      * ------------- JFreeChart 1.0.x ---------------------------------------------
56      * 11-Jan-2006 : Added new method expandToInclude(Range, double) (DG);
57      * 18-Dec-2007 : New methods intersects(Range) and scale(...) thanks to Sergei
58      *               Ivanov (DG);
59      * 08-Jan-2012 : New method combineIgnoringNaN() (DG);
60      * 23-Feb-2014 : Added isNaNRange() method (DG);
61      *
62      */
63
64    package org.jfree.data;
65
66    import java.io.Serializable;
67    import org.jfree.chart.util.ParamChecks;
68
69    /**
70     * Represents an immutable range of values.
71     */
72    public strictfp class Range implements Serializable {
73
74        /** For serialization. */
75        private static final long serialVersionUID = -906333695431863380L;
76
77        /** The lower bound of the range. */
78        private double lower;
79
80        /** The upper bound of the range. */
81        private double upper;
82
83        /**
84         * Creates a new range.
85         *
86         * @param lower  the lower bound (must be &lt;= upper bound).
87         * @param upper  the upper bound (must be &gt;= lower bound).
88         */
89        public Range(double lower, double upper) {
90  4         if (lower > upper) {
91  3             String msg = "Range(double, double): require lower (" + lower
92  3                 + ") <= upper (" + upper + ").";
93  1             throw new IllegalArgumentException(msg);
94            }
95  1         this.lower = lower;
96  1         this.upper = upper;
97        }
98
99        /**
100        * Returns the lower bound for the range.
101        *
102        * @return The lower bound.
103        */
104       public double getLowerBound() {
105 1         return this.lower;
106       }
107
108       /**
109        * Returns the upper bound for the range.
110        *
111        * @return The upper bound.
```

```
112         */
113        public double getUpperBound() {
114 1          return this.upper;
115        }
116
117        /**
118         * Returns the length of the range.
119         *
120         * @return The length.
121         */
122        public double getLength() {
123 2          return this.upper - this.lower;
124        }
125
126        /**
127         * Returns the central value for the range.
128         *
129         * @return The central value.
130         */
131        public double getCentralValue() {
132 6          return this.lower / 2.0 + this.upper / 2.0;
133        }
134
135        /**
136         * Returns <code>true</code> if the range contains the specified value and
137         * <code>false</code> otherwise.
138         *
139         * @param value  the value to lookup.
140         *
141         * @return <code>true</code> if the range contains the specified value.
142         */
143        public boolean contains(double value) {
144 12          return (value >= this.lower && value <= this.upper);
145        }
146
147        /**
148         * Returns <code>true</code> if the range intersects with the specified
149         * range, and <code>false</code> otherwise.
150         *
151         * @param b0  the lower bound (should be &lt;= b1).
152         * @param b1  the upper bound (should be &gt;= b0).
153         *
154         * @return A boolean.
155         */
156        public boolean intersects(double b0, double b1) {
157 4          if (b0 <= this.lower) {
158 8              return (b1 > this.lower);
159            }
160            else {
161 12              return (b0 < this.upper && b1 >= b0);
162            }
163        }
164
165        /**
166         * Returns <code>true</code> if the range intersects with the specified
167         * range, and <code>false</code> otherwise.
168         *
```

```
169         * @param range  another range (<code>null</code> not permitted).
170         *
171         * @return A boolean.
172         *
173         * @since 1.0.9
174         */
175        public boolean intersects(Range range) {
176 4          return intersects(range.getLowerBound(), range.getUpperBound());
177        }
178
179        /**
180         * Returns the value within the range that is closest to the specified
181         * value.
182         *
183         * @param value  the value.
184         *
185         * @return The constrained value.
186         */
187        public double constrain(double value) {
188            double result = value;
189 4          if (!contains(value)) {
190 4              if (value > this.upper) {
191                    result = this.upper;
192                }
193 4              else if (value < this.lower) {
194                    result = this.lower;
195                }
196            }
197 1          return result;
198        }
199
200        /**
201         * Creates a new range by combining two existing ranges.
202         * <P>
203         * Note that:
204         * <ul>
205         *   <li>either range can be <code>null</code>, in which case the other
206         *       range is returned;</li>
207         *   <li>if both ranges are <code>null</code> the return value is
208         *       <code>null</code>.</li>
209         * </ul>
210         *
211         * @param range1  the first range (<code>null</code> permitted).
212         * @param range2  the second range (<code>null</code> permitted).
213         *
214         * @return A new range (possibly <code>null</code>).
215         */
216        public static Range combine(Range range1, Range range2) {
217 3          if (range1 == null) {
218 1              return range2;
219            }
220 3          if (range2 == null) {
221 1              return range1;
222            }
223 4          double l = Math.min(range1.getLowerBound(), range2.getLowerBound());
224 4          double u = Math.max(range1.getUpperBound(), range2.getUpperBound());
225 2          return new Range(l, u);
```

```
226       }
227
228       /**
229        * Returns a new range that spans both <code>range1</code> and
230        * <code>range2</code>.  This method has a special handling to ignore
231        * Double.NaN values.
232        *
233        * @param range1  the first range (<code>null</code> permitted).
234        * @param range2  the second range (<code>null</code> permitted).
235        *
236        * @return A new range (possibly <code>null</code>).
237        *
238        * @since 1.0.15
239        */
240       public static Range combineIgnoringNaN(Range range1, Range range2) {
241 3         if (range1 == null) {
242 7             if (range2 != null && range2.isNaNRange()) {
243 1                 return null;
244               }
245 1             return range2;
246           }
247 3         if (range2 == null) {
248 4             if (range1.isNaNRange()) {
249 1                 return null;
250               }
251 1             return range1;
252           }
253 4         double l = min(range1.getLowerBound(), range2.getLowerBound());
254 4         double u = max(range1.getUpperBound(), range2.getUpperBound());
255 8         if (Double.isNaN(l) && Double.isNaN(u)) {
256 1             return null;
257           }
258 2         return new Range(l, u);
259       }
260
261       /**
262        * Returns the minimum value.  If either value is NaN, the other value is
263        * returned.  If both are NaN, NaN is returned.
264        *
265        * @param d1  value 1.
266        * @param d2  value 2.
267        *
268        * @return The minimum of the two values.
269        */
270       private static double min(double d1, double d2) {
271 4         if (Double.isNaN(d1)) {
272 1             return d2;
273           }
274 4         if (Double.isNaN(d2)) {
275 1             return d1;
276           }
277 3         return Math.min(d1, d2);
278       }
279
280       private static double max(double d1, double d2) {
281 4         if (Double.isNaN(d1)) {
282 1             return d2;
```

```
283           }
284 4         if (Double.isNaN(d2)) {
285 1             return d1;
286           }
287 3         return Math.max(d1, d2);
288       }
289
290       /**
291        * Returns a range that includes all the values in the specified
292        * <code>range</code> AND the specified <code>value</code>.
293        *
294        * @param range  the range (<code>null</code> permitted).
295        * @param value  the value that must be included.
296        *
297        * @return A range.
298        *
299        * @since 1.0.1
300        */
301       public static Range expandToInclude(Range range, double value) {
302 3         if (range == null) {
303 2             return new Range(value, value);
304           }
305 5         if (value < range.getLowerBound()) {
306 3             return new Range(value, range.getUpperBound());
307           }
308 5         else if (value > range.getUpperBound()) {
309 3             return new Range(range.getLowerBound(), value);
310           }
311           else {
312 1             return range;
313           }
314       }
315
316       /**
317        * Creates a new range by adding margins to an existing range.
318        *
319        * @param range  the range (<code>null</code> not permitted).
320        * @param lowerMargin  the lower margin (expressed as a percentage of the
321        *                     range length).
322        * @param upperMargin  the upper margin (expressed as a percentage of the
323        *                     range length).
324        *
325        * @return The expanded range.
326        */
327       public static Range expand(Range range,
328                                  double lowerMargin, double upperMargin) {
329 1         ParamChecks.nullNotPermitted(range, "range");
330 1         double length = range.getLength();
331 3         double lower = range.getLowerBound() - length * lowerMargin;
332 3         double upper = range.getUpperBound() + length * upperMargin;
333 4         if (lower > upper) {
334 5             lower = lower / 2.0 + upper / 2.0;
335               upper = lower;
336           }
337 2         return new Range(lower, upper);
338       }
339
```

```
340       /**
341        * Shifts the range by the specified amount.
342        *
343        * @param base  the base range (<code>null</code> not permitted).
344        * @param delta  the shift amount.
345        *
346        * @return A new range.
347        */
348       public static Range shift(Range base, double delta) {
349 4         return shift(base, delta, false);
350       }
351
352       /**
353        * Shifts the range by the specified amount.
354        *
355        * @param base  the base range (<code>null</code> not permitted).
356        * @param delta  the shift amount.
357        * @param allowZeroCrossing  a flag that determines whether or not the
358        *                           bounds of the range are allowed to cross
359        *                           zero after adjustment.
360        *
361        * @return A new range.
362        */
363       public static Range shift(Range base, double delta,
364                                 boolean allowZeroCrossing) {
365 1         ParamChecks.nullNotPermitted(base, "base");
366 3         if (allowZeroCrossing) {
367 4             return new Range(base.getLowerBound() + delta,
368 2                     base.getUpperBound() + delta);
369           }
370           else {
371 5             return new Range(shiftWithNoZeroCrossing(base.getLowerBound(),
372 3                     delta), shiftWithNoZeroCrossing(base.getUpperBound(),
373                       delta));
374           }
375       }
376
377       /**
378        * Returns the given <code>value</code> adjusted by <code>delta</code> but
379        * with a check to prevent the result from crossing <code>0.0</code>.
380        *
381        * @param value  the value.
382        * @param delta  the adjustment.
383        *
384        * @return The adjusted value.
385        */
386       private static double shiftWithNoZeroCrossing(double value, double delta) {
387 5         if (value > 0.0) {
388 5             return Math.max(value + delta, 0.0);
389           }
390 5         else if (value < 0.0) {
391 5             return Math.min(value + delta, 0.0);
392           }
393           else {
394 2             return value + delta;
395           }
396       }
```

```
397
398       /**
399        * Scales the range by the specified factor.
400        *
401        * @param base the base range (<code>null</code> not permitted).
402        * @param factor the scaling factor (must be non-negative).
403        *
404        * @return A new range.
405        *
406        * @since 1.0.9
407        */
408       public static Range scale(Range base, double factor) {
409 1         ParamChecks.nullNotPermitted(base, "base");
410 5         if (factor < 0) {
411 1             throw new IllegalArgumentException("Negative 'factor' argument.");
412           }
413 4         return new Range(base.getLowerBound() * factor,
414 2                 base.getUpperBound() * factor);
415       }
416
417       /**
418        * Tests this object for equality with an arbitrary object.
419        *
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 3         if (!(obj instanceof Range)) {
427 2             return false;
428           }
429           Range range = (Range) obj;
430 3         if (!(this.lower == range.lower)) {
431 2             return false;
432           }
433 3         if (!(this.upper == range.upper)) {
434 2             return false;
435           }
436 2         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 12        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 1          temp = Double.doubleToLongBits(this.lower);
461 3          result = (int) (temp ^ (temp >>> 32));
462 1          temp = Double.doubleToLongBits(this.upper);
463 6          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 7          return ("Range[" + this.lower + "," + this.upper + "]");
476        }
477
478    }
```

Mutations

90	1. changed conditional boundary → KILLED 2. negated conditional → KILLED 3. removed conditional - replaced comparison check with false → SURVIVED 4. removed conditional - replaced comparison check with true → KILLED
91	1. removed call to java/lang/StringBuilder::<init> → NO_COVERAGE 2. removed call to java/lang/StringBuilder::append → NO_COVERAGE 3. removed call to java/lang/StringBuilder::toString → NO_COVERAGE
92	1. removed call to java/lang/StringBuilder::append → NO_COVERAGE 2. removed call to java/lang/StringBuilder::append → NO_COVERAGE 3. removed call to java/lang/StringBuilder::append → NO_COVERAGE
93	1. removed call to java/lang/IllegalArgumentException::<init> → NO_COVERAGE
95	1. Removed assignment to member variable lower → KILLED
96	1. Removed assignment to member variable upper → KILLED
105	1. replaced return of double value with -(x + 1) for org/jfree/data/Range::getLowerBound → KILLED
114	1. replaced return of double value with -(x + 1) for org/jfree/data/Range::getUpperBound → KILLED
123	1. Replaced double subtraction with addition → KILLED 2. replaced return of double value with -(x + 1) for org/jfree/data/Range::getLength → KILLED
132	1. Substituted 2.0 with 1.0 → KILLED 2. Substituted 2.0 with 1.0 → KILLED 3. Replaced double division with multiplication → KILLED 4. Replaced double division with multiplication → KILLED 5. Replaced double addition with subtraction → KILLED 6. replaced return of double value with -(x + 1) for org/jfree/data/Range::getCentralValue → KILLED
144	1. changed conditional boundary → SURVIVED 2. changed conditional boundary → SURVIVED 3. Substituted 1 with 0 → KILLED 4. Substituted 0 with 1 → KILLED 5. negated conditional → KILLED 6. negated conditional → KILLED 7. removed conditional - replaced comparison check with false → KILLED 8. removed conditional - replaced comparison check with false → KILLED 9. removed conditional - replaced comparison check with true → KILLED 10. removed conditional - replaced comparison check with true → KILLED

	11. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED 12. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED
157	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED 3. removed conditional - replaced comparison check with false → KILLED 4. removed conditional - replaced comparison check with true → KILLED
158	1. changed conditional boundary → SURVIVED 2. Substituted 1 with 0 → KILLED 3. Substituted 0 with 1 → KILLED 4. negated conditional → KILLED 5. removed conditional - replaced comparison check with false → KILLED 6. removed conditional - replaced comparison check with true → KILLED 7. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED 8. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED
161	1. changed conditional boundary → SURVIVED 2. changed conditional boundary → KILLED 3. Substituted 1 with 0 → KILLED 4. Substituted 0 with 1 → KILLED 5. negated conditional → KILLED 6. negated conditional → KILLED 7. removed conditional - replaced comparison check with false → KILLED 8. removed conditional - replaced comparison check with false → KILLED 9. removed conditional - replaced comparison check with true → SURVIVED 10. removed conditional - replaced comparison check with true → KILLED 11. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED 12. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED
176	1. removed call to org/jfree/data/Range::getLowerBound → SURVIVED 2. removed call to org/jfree/data/Range::getUpperBound → KILLED 3. removed call to org/jfree/data/Range::intersects → KILLED 4. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED
189	1. negated conditional → KILLED 2. removed call to org/jfree/data/Range::contains → SURVIVED 3. removed conditional - replaced equality check with false → KILLED 4. removed conditional - replaced equality check with true → SURVIVED
190	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED 3. removed conditional - replaced comparison check with false → KILLED 4. removed conditional - replaced comparison check with true → KILLED
193	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED 3. removed conditional - replaced comparison check with false → KILLED 4. removed conditional - replaced comparison check with true → SURVIVED
197	1. replaced return of double value with -(x + 1) for org/jfree/data/Range::constrain → KILLED
217	1. negated conditional → KILLED 2. removed conditional - replaced equality check with false → KILLED 3. removed conditional - replaced equality check with true → KILLED
218	1. mutated return of Object value for org/jfree/data/Range::combine to (if (x != null) null else throw new RuntimeException) → KILLED
220	1. negated conditional → KILLED 2. removed conditional - replaced equality check with false → KILLED 3. removed conditional - replaced equality check with true → SURVIVED
221	1. mutated return of Object value for org/jfree/data/Range::combine to (if (x != null) null else throw new RuntimeException) → KILLED
223	1. replaced call to java/lang/Math::min with argument → KILLED 2. removed call to org/jfree/data/Range::getLowerBound → KILLED 3. removed call to org/jfree/data/Range::getLowerBound → KILLED 4. removed call to java/lang/Math::min → KILLED
224	1. replaced call to java/lang/Math::max with argument → KILLED 2. removed call to org/jfree/data/Range::getUpperBound → KILLED 3. removed call to org/jfree/data/Range::getUpperBound → SURVIVED 4. removed call to java/lang/Math::max → KILLED
225	1. removed call to org/jfree/data/Range::<init> → KILLED 2. mutated return of Object value for org/jfree/data/Range::combine to (if (x != null) null else throw new RuntimeException) → KILLED
241	1. negated conditional → KILLED

	2. removed conditional - replaced equality check with false → KILLED 3. removed conditional - replaced equality check with true → KILLED
242	1. negated conditional → KILLED 2. negated conditional → KILLED 3. removed call to org/jfree/data/Range::isNaNRange → KILLED 4. removed conditional - replaced equality check with false → KILLED 5. removed conditional - replaced equality check with false → KILLED 6. removed conditional - replaced equality check with true → KILLED 7. removed conditional - replaced equality check with true → KILLED
243	1. mutated return of Object value for org/jfree/data/Range::combineIgnoringNaN to (if (x != null) null else throw new RuntimeException) → KILLED
245	1. mutated return of Object value for org/jfree/data/Range::combineIgnoringNaN to (if (x != null) null else throw new RuntimeException) → KILLED
247	1. negated conditional → KILLED 2. removed conditional - replaced equality check with false → KILLED 3. removed conditional - replaced equality check with true → SURVIVED
248	1. negated conditional → KILLED 2. removed call to org/jfree/data/Range::isNaNRange → KILLED 3. removed conditional - replaced equality check with false → KILLED 4. removed conditional - replaced equality check with true → KILLED
249	1. mutated return of Object value for org/jfree/data/Range::combineIgnoringNaN to (if (x != null) null else throw new RuntimeException) → KILLED
251	1. mutated return of Object value for org/jfree/data/Range::combineIgnoringNaN to (if (x != null) null else throw new RuntimeException) → KILLED
253	1. replaced call to org/jfree/data/Range::min with argument → KILLED 2. removed call to org/jfree/data/Range::getLowerBound → KILLED 3. removed call to org/jfree/data/Range::getLowerBound → KILLED 4. removed call to org/jfree/data/Range::min → KILLED
254	1. replaced call to org/jfree/data/Range::max with argument → KILLED 2. removed call to org/jfree/data/Range::getUpperBound → KILLED 3. removed call to org/jfree/data/Range::getUpperBound → KILLED 4. removed call to org/jfree/data/Range::max → KILLED
255	1. negated conditional → KILLED 2. negated conditional → KILLED 3. removed call to java/lang/Double::isNaN → KILLED 4. removed call to java/lang/Double::isNaN → KILLED 5. removed conditional - replaced equality check with false → KILLED 6. removed conditional - replaced equality check with false → KILLED 7. removed conditional - replaced equality check with true → SURVIVED 8. removed conditional - replaced equality check with true → SURVIVED
256	1. mutated return of Object value for org/jfree/data/Range::combineIgnoringNaN to (if (x != null) null else throw new RuntimeException) → KILLED
258	1. removed call to org/jfree/data/Range::<init> → KILLED 2. mutated return of Object value for org/jfree/data/Range::combineIgnoringNaN to (if (x != null) null else throw new RuntimeException) → KILLED
271	1. negated conditional → KILLED 2. removed call to java/lang/Double::isNaN → SURVIVED 3. removed conditional - replaced equality check with false → SURVIVED 4. removed conditional - replaced equality check with true → KILLED
272	1. replaced return of double value with -(x + 1) for org/jfree/data/Range::min → KILLED
274	1. negated conditional → SURVIVED 2. removed call to java/lang/Double::isNaN → SURVIVED 3. removed conditional - replaced equality check with false → SURVIVED 4. removed conditional - replaced equality check with true → SURVIVED
275	1. replaced return of double value with -(x + 1) for org/jfree/data/Range::min → NO_COVERAGE
277	1. replaced call to java/lang/Math::min with argument → KILLED 2. removed call to java/lang/Math::min → KILLED 3. replaced return of double value with -(x + 1) for org/jfree/data/Range::min → KILLED
281	1. negated conditional → KILLED 2. removed call to java/lang/Double::isNaN → SURVIVED 3. removed conditional - replaced equality check with false → SURVIVED 4. removed conditional - replaced equality check with true → KILLED
282	1. replaced return of double value with -(x + 1) for org/jfree/data/Range::max → KILLED
284	1. negated conditional → SURVIVED

	2. removed call to java/lang/Double::isNaN → SURVIVED 3. removed conditional - replaced equality check with false → SURVIVED 4. removed conditional - replaced equality check with true → SURVIVED
285	1. replaced return of double value with -(x + 1) for org/jfree/data/Range::max → NO_COVERAGE
287	1. replaced call to java/lang/Math::max with argument → KILLED 2. removed call to java/lang/Math::max → KILLED 3. replaced return of double value with -(x + 1) for org/jfree/data/Range::max → KILLED
302	1. negated conditional → KILLED 2. removed conditional - replaced equality check with false → KILLED 3. removed conditional - replaced equality check with true → KILLED
303	1. removed call to org/jfree/data/Range::<init> → KILLED 2. mutated return of Object value for org/jfree/data/Range::expandToInclude to (if (x != null) null else throw new RuntimeException) → KILLED
305	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED 3. removed call to org/jfree/data/Range::getLowerBound → KILLED 4. removed conditional - replaced comparison check with false → KILLED 5. removed conditional - replaced comparison check with true → KILLED
306	1. removed call to org/jfree/data/Range::<init> → KILLED 2. removed call to org/jfree/data/Range::getUpperBound → KILLED 3. mutated return of Object value for org/jfree/data/Range::expandToInclude to (if (x != null) null else throw new RuntimeException) → KILLED
308	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED 3. removed call to org/jfree/data/Range::getUpperBound → SURVIVED 4. removed conditional - replaced comparison check with false → KILLED 5. removed conditional - replaced comparison check with true → SURVIVED
309	1. removed call to org/jfree/data/Range::<init> → KILLED 2. removed call to org/jfree/data/Range::getLowerBound → KILLED 3. mutated return of Object value for org/jfree/data/Range::expandToInclude to (if (x != null) null else throw new RuntimeException) → KILLED
312	1. mutated return of Object value for org/jfree/data/Range::expandToInclude to (if (x != null) null else throw new RuntimeException) → KILLED
329	1. removed call to org/jfree/chart/util/ParamChecks::nullNotPermitted → SURVIVED
330	1. removed call to org/jfree/data/Range::getLength → KILLED
331	1. Replaced double multiplication with division → KILLED 2. Replaced double subtraction with addition → KILLED 3. removed call to org/jfree/data/Range::getLowerBound → KILLED
332	1. Replaced double multiplication with division → KILLED 2. Replaced double addition with subtraction → KILLED 3. removed call to org/jfree/data/Range::getUpperBound → KILLED
333	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED 3. removed conditional - replaced comparison check with false → KILLED 4. removed conditional - replaced comparison check with true → KILLED
334	1. Substituted 2.0 with 1.0 → KILLED 2. Substituted 2.0 with 1.0 → KILLED 3. Replaced double division with multiplication → KILLED 4. Replaced double division with multiplication → KILLED 5. Replaced double addition with subtraction → KILLED
337	1. removed call to org/jfree/data/Range::<init> → KILLED 2. mutated return of Object value for org/jfree/data/Range::expand to (if (x != null) null else throw new RuntimeException) → KILLED
349	1. replaced call to org/jfree/data/Range::shift with argument → KILLED 2. Substituted 0 with 1 → SURVIVED 3. removed call to org/jfree/data/Range::shift → KILLED 4. mutated return of Object value for org/jfree/data/Range::shift to (if (x != null) null else throw new RuntimeException) → KILLED
365	1. removed call to org/jfree/chart/util/ParamChecks::nullNotPermitted → SURVIVED
366	1. negated conditional → SURVIVED 2. removed conditional - replaced equality check with false → SURVIVED 3. removed conditional - replaced equality check with true → SURVIVED
367	1. removed call to org/jfree/data/Range::<init> → KILLED 2. Replaced double addition with subtraction → SURVIVED

	3. removed call to org/jfree/data/Range::getLowerBound → KILLED 4. mutated return of Object value for org/jfree/data/Range::shift to (if (x != null) null else throw new RuntimeException) → KILLED
368	1. Replaced double addition with subtraction → SURVIVED 2. removed call to org/jfree/data/Range::getUpperBound → KILLED
371	1. replaced call to org/jfree/data/Range::shiftWithNoZeroCrossing with argument → KILLED 2. removed call to org/jfree/data/Range::<init> → KILLED 3. removed call to org/jfree/data/Range::getLowerBound → KILLED 4. removed call to org/jfree/data/Range::shiftWithNoZeroCrossing → KILLED 5. mutated return of Object value for org/jfree/data/Range::shift to (if (x != null) null else throw new RuntimeException) → KILLED
372	1. replaced call to org/jfree/data/Range::shiftWithNoZeroCrossing with argument → KILLED 2. removed call to org/jfree/data/Range::getUpperBound → KILLED 3. removed call to org/jfree/data/Range::shiftWithNoZeroCrossing → KILLED
387	1. changed conditional boundary → SURVIVED 2. Substituted 0.0 with 1.0 → SURVIVED 3. negated conditional → SURVIVED 4. removed conditional - replaced comparison check with false → SURVIVED 5. removed conditional - replaced comparison check with true → SURVIVED
388	1. replaced call to java/lang/Math::max with argument → KILLED 2. Substituted 0.0 with 1.0 → SURVIVED 3. Replaced double addition with subtraction → KILLED 4. removed call to java/lang/Math::max → KILLED 5. replaced return of double value with -(x + 1) for org/jfree/data/Range::shiftWithNoZeroCrossing → KILLED
390	1. changed conditional boundary → NO_COVERAGE 2. Substituted 0.0 with 1.0 → NO_COVERAGE 3. negated conditional → NO_COVERAGE 4. removed conditional - replaced comparison check with false → NO_COVERAGE 5. removed conditional - replaced comparison check with true → NO_COVERAGE
391	1. replaced call to java/lang/Math::min with argument → NO_COVERAGE 2. Substituted 0.0 with 1.0 → NO_COVERAGE 3. Replaced double addition with subtraction → NO_COVERAGE 4. removed call to java/lang/Math::min → NO_COVERAGE 5. replaced return of double value with -(x + 1) for org/jfree/data/Range::shiftWithNoZeroCrossing → NO_COVERAGE
394	1. Replaced double addition with subtraction → NO_COVERAGE 2. replaced return of double value with -(x + 1) for org/jfree/data/Range::shiftWithNoZeroCrossing → NO_COVERAGE
409	1. removed call to org/jfree/chart/util/ParamChecks::nullNotPermitted → SURVIVED
410	1. changed conditional boundary → SURVIVED 2. Substituted 0.0 with 1.0 → SURVIVED 3. negated conditional → SURVIVED 4. removed conditional - replaced comparison check with false → SURVIVED 5. removed conditional - replaced comparison check with true → SURVIVED
411	1. removed call to java/lang/IllegalArgumentException::<init> → SURVIVED
413	1. removed call to org/jfree/data/Range::<init> → NO_COVERAGE 2. Replaced double multiplication with division → NO_COVERAGE 3. removed call to org/jfree/data/Range::getLowerBound → NO_COVERAGE 4. mutated return of Object value for org/jfree/data/Range::scale to (if (x != null) null else throw new RuntimeException) → NO_COVERAGE
414	1. Replaced double multiplication with division → NO_COVERAGE 2. removed call to org/jfree/data/Range::getUpperBound → NO_COVERAGE
426	1. negated conditional → KILLED 2. removed conditional - replaced equality check with false → KILLED 3. removed conditional - replaced equality check with true → KILLED
427	1. Substituted 0 with 1 → KILLED 2. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED
430	1. negated conditional → KILLED 2. removed conditional - replaced equality check with false → KILLED 3. removed conditional - replaced equality check with true → KILLED
431	1. Substituted 0 with 1 → KILLED 2. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED
433	1. negated conditional → KILLED 2. removed conditional - replaced equality check with false → KILLED

	3. removed conditional - replaced equality check with true → KILLED
434	1. Substituted 0 with 1 → KILLED 2. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED
436	1. Substituted 1 with 0 → KILLED 2. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED
448	1. Substituted 1 with 0 → KILLED 2. Substituted 0 with 1 → KILLED 3. negated conditional → KILLED 4. negated conditional → KILLED 5. removed call to java/lang/Double::isNaN → KILLED 6. removed call to java/lang/Double::isNaN → KILLED 7. removed conditional - replaced equality check with false → KILLED 8. removed conditional - replaced equality check with false → KILLED 9. removed conditional - replaced equality check with true → SURVIVED 10. removed conditional - replaced equality check with true → SURVIVED 11. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED 12. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED
460	1. removed call to java/lang/Double::doubleToLongBits → KILLED
461	1. Substituted 32 with 33 → KILLED 2. Replaced Unsigned Shift Right with Shift Left → KILLED 3. Replaced XOR with AND → KILLED
462	1. removed call to java/lang/Double::doubleToLongBits → KILLED
463	1. Substituted 29 with 30 → KILLED 2. Substituted 32 with 33 → KILLED 3. Replaced integer multiplication with division → KILLED 4. Replaced Unsigned Shift Right with Shift Left → KILLED 5. Replaced XOR with AND → KILLED 6. Replaced integer addition with subtraction → KILLED
464	1. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED
475	1. removed call to java/lang/StringBuilder::<init> → NO_COVERAGE 2. removed call to java/lang/StringBuilder::append → NO_COVERAGE 3. removed call to java/lang/StringBuilder::append → NO_COVERAGE 4. removed call to java/lang/StringBuilder::append → NO_COVERAGE 5. removed call to java/lang/StringBuilder::append → NO_COVERAGE 6. removed call to java/lang/StringBuilder::toString → NO_COVERAGE 7. mutated return of Object value for org/jfree/data/Range::toString to (if (x != null) null else throw new RuntimeException) → NO_COVERAGE

Active mutators

- RETURN_VALS_MUTATOR
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_61
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_60
- CONDITIONALS_BOUNDARY_MUTATOR
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_56
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_55
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_58
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_57
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_52
- VOID_METHOD_CALL_MUTATOR
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_51
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_54
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_53
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_59
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_50
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_45
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_44
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_47
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_46
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_41
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_40
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_43
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_42
- NEGATE_CONDITIONALS_MUTATOR
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_49
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_48
- INLINE_CONSTANT_MUTATOR
- CONSTRUCTOR_CALL_MUTATOR
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_34
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_33
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_36

- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_35
- EXPERIMENTAL_MEMBER_VARIABLE_MUTATOR
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_30
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_32
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_31
- REMOVE_CONDITIONALS_ORDER_ELSE_MUTATOR
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_38
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_37
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_39
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_3
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_2
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_1
- INVERT_NEGS_MUTATOR
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_0
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_23
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_22
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_25
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_9
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_24
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_8
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_7
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_6
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_21
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_5
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_20
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_4
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_27
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_26
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_29
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_28
- REMOVE_INCREMENTS_MUTATOR
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_12
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_11
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_99
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_14
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_13
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_96
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_95
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_10
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_98
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_97
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_19
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_16
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_15
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_18
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_17
- EXPERIMENTAL_SWITCH_MUTATOR
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_92
- ARGUMENT_PROPAGATION_MUTATOR
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_91
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_94
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_93
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_90
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_89
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_88
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_85
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_84
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_87
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_86
- MATH_MUTATOR
- NON_VOID_METHOD_CALL_MUTATOR
- REMOVE_CONDITIONALS_EQUAL_IF_MUTATOR
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_81
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_80
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_83
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_82
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_78
- REMOVE_CONDITIONALS_EQUAL_ELSE_MUTATOR
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_77
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_79
- INCREMENTS_MUTATOR
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_74
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_73
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_76
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_75
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_70
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_72
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_71
- REMOVE_CONDITIONALS_ORDER_IF_MUTATOR
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_67
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_66

- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_69
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_68
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_63
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_62
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_65
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_64

Tests examined

- org.jfree.data.test.GetLowerBoundRange.testLowerBoundBothLimitNegative(org.jfree.data.test.GetLowerBoundRange) (0 ms)
- org.jfree.data.test.RangeCombineIgnoringNaNTest.combineBothParamsNull(org.jfree.data.test.RangeCombineIgnoringNaNTest) (0 ms)
- org.jfree.data.test.RangeExpandToIncludeTest.lessThanLower(org.jfree.data.test.RangeExpandToIncludeTest) (0 ms)
- org.jfree.data.test.RangeCombineTest.combineBothParamsNull(org.jfree.data.test.RangeCombineTest) (0 ms)
- org.jfree.data.test.RangeCombineTest.combineNoNullParams(org.jfree.data.test.RangeCombineTest) (1 ms)
- org.jfree.data.test.GetLowerBoundRange.testLowerBoundUpperLimitPositive(org.jfree.data.test.GetLowerBoundRange) (0 ms)
- org.jfree.data.test.GetUpperBoundRange.testUpperBoundLowerNegative(org.jfree.data.test.GetUpperBoundRange) (0 ms)
- org.jfree.data.test.RangeExpandToIncludeTest.rangeIsNull(org.jfree.data.test.RangeExpandToIncludeTest) (1 ms)
- org.jfree.data.test.RangeEqualsTest.equalsRange2IsNotARange(org.jfree.data.test.RangeEqualsTest) (1 ms)
- org.jfree.data.test.RangeExpandTest.RangeAppropriateValuesTest(org.jfree.data.test.RangeExpandTest) (0 ms)
- org.jfree.data.test.RangeCombineIgnoringNaNTest.combineRange2IsNullAndNotNaN(org.jfree.data.test.RangeCombineIgnoringNaNTest) (0 ms)
- org.jfree.data.test.RangeCombineIgnoringNaNTest.combineBothParamsMakeANaNResult(org.jfree.data.test.RangeCombineIgnoringNaNTest) (1 ms)
- org.jfree.data.test.RangeIntersectsTest.intersectsWithinBoundsWhereUpperGreaterThanLowerTest(org.jfree.data.test.RangeIntersectsTest) (0 ms)
- org.jfree.data.test.RangeShiftTest.shiftAllowZeroCrossingGivenZero(org.jfree.data.test.RangeShiftTest) (0 ms)
- org.jfree.data.test.RangeIntersectsTest.intersectsOutsideBoundsWhereUpperEqualsLowerTest(org.jfree.data.test.RangeIntersectsTest) (0 ms)
- org.jfree.data.test.RangeShiftTest.shiftBaseRangeIsNull(org.jfree.data.test.RangeShiftTest) (1 ms)
- org.jfree.data.test.RangeIntersectsTest.intersectsWithUpperLessThanLowerButInvalidTest(org.jfree.data.test.RangeIntersectsTest) (0 ms)
- org.jfree.data.test.RangeCombineIgnoringNaNTest.combineRange1IsNull(org.jfree.data.test.RangeCombineIgnoringNaNTest) (0 ms)
- org.jfree.data.test.GetLowerBoundRange.testLowerBound(org.jfree.data.test.GetLowerBoundRange) (1 ms)
- org.jfree.data.test.RangeIntersectsTest.intersectsWithRangeObjectTest(org.jfree.data.test.RangeIntersectsTest) (1 ms)
- org.jfree.data.test.RangeIntersectsTest.intersectsWithinBoundsWhereUpperEqualsLowerTest(org.jfree.data.test.RangeIntersectsTest) (1 ms)
- org.jfree.data.test.RangeCombineTest.combineRange1IsNull(org.jfree.data.test.RangeCombineTest) (0 ms)
- org.jfree.data.test.RangeConstrainTest.constrainValueLessThanLowerBound(org.jfree.data.test.RangeConstrainTest) (1 ms)
- org.jfree.data.test.RangeShiftTest.shiftNotAllowingZeroCrossingWithDeltaNotEqualZero(org.jfree.data.test.RangeShiftTest) (0 ms)
- org.jfree.data.test.Range_containsTest.contains_trueTest(org.jfree.data.test.Range_containsTest) (0 ms)
- org.jfree.data.test.RangeExpandTest.RangeLowerRangeGreaterTest(org.jfree.data.test.RangeExpandTest) (0 ms)
- org.jfree.data.test.RangeExpandToIncludeTest.rangeIsNotNull(org.jfree.data.test.RangeExpandToIncludeTest) (0 ms)
- org.jfree.data.test.RangeCombineIgnoringNaNTest.combineRange1IsNullAndRange2IsNaN(org.jfree.data.test.RangeCombineIgnoringNaNTest) (1 ms)
- org.jfree.data.test.RangeCombineIgnoringNaNTest.combineRange2IsNullAndRange1IsNaN(org.jfree.data.test.RangeCombineIgnoringNaNTest) (1 ms)
- org.jfree.data.test.GetUpperBoundRange.testUpperBoundBothNegative(org.jfree.data.test.GetUpperBoundRange) (0 ms)
- org.jfree.data.test.GetUpperBoundRange.testUpperBoundBothPositive(org.jfree.data.test.GetUpperBoundRange) (1 ms)
- org.jfree.data.test.Range_containsTest.contains_falseTest(org.jfree.data.test.Range_containsTest) (0 ms)
- org.jfree.data.test.RangeExpandToIncludeTest.moreThanUpper(org.jfree.data.test.RangeExpandToIncludeTest) (0 ms)
- org.jfree.data.test.RangeEqualsTest.equalsUpperBoundsNotEqaul(org.jfree.data.test.RangeEqualsTest) (0 ms)
- org.jfree.data.test.RangeConstrainTest.constrainValueWithinBound(org.jfree.data.test.RangeConstrainTest) (0 ms)
- org.jfree.data.test.RangeConstrainTest.constrainValueGreaterThanUpperBound(org.jfree.data.test.RangeConstrainTest) (0 ms)
- org.jfree.data.test.RangeIntersectsTest.intersectsOutsideBoundsWhereUpperGreaterThanLowerTest(org.jfree.data.test.RangeIntersectsTest) (0 ms)
- org.jfree.data.test.RangeCombineTest.combineRange2IsNull(org.jfree.data.test.RangeCombineTest) (1 ms)
- org.jfree.data.test.RangeScaleTest.scaleFactorLessThanZero(org.jfree.data.test.RangeScaleTest) (1 ms)
- org.jfree.data.test.RangeIntersectsTest.intersectsOutsideBoundsWhereUpperLessThanLowerTest(org.jfree.data.test.RangeIntersectsTest) (0 ms)
- org.jfree.data.test.RangeHashCodeTest.RangeGetCorrectHashCodeTest(org.jfree.data.test.RangeHashCodeTest) (3 ms)
- org.jfree.data.test.RangeEqualsTest.equalsLowerBoundsNotEqaul(org.jfree.data.test.RangeEqualsTest) (0 ms)
- org.jfree.data.test.RangeIntersectsTest.intersectsWithinBoundsWhereUpperLessThanLowerTest(org.jfree.data.test.RangeIntersectsTest) (0 ms)
- org.jfree.data.test.RangeGetCentralValueTest.getCentralValueCorrectlyTest(org.jfree.data.test.RangeGetCentralValueTest) (0 ms)
- org.jfree.data.test.RangeCombineIgnoringNaNTest.combineNoNullParams(org.jfree.data.test.RangeCombineIgnoringNaNTest) (1 ms)

Report generated by PIT 1.1.9