

# Rectangle Test Report

Team members:

Mario LoPrinzi  
&  
Christina Xuan Yu

## 1. Test Plan:

- Test all the methods, to see if they function as described.
- We focus on some extreme cases, such as testing with negative numbers, randomly generated numbers, numbers that are out of bounds.
- Make sure the program fails where it should.
  - Make sure a Rectangle cannot be made with negative width or height
  - Make sure the area cannot overflow an integer
  - Make sure a Rectangle with no area cannot contain any points

## 2. Positive Test: Christina Yu: 5, Mario LoPrinzi: 3

- testArea(): test rectangle area starting from random point with random width and height
- testDiagonal(): test diagonal using random width and height
- testUnion(): test the rectangle union with empty
- testIntersects(): test if the rectangle intersects with itself
- testIntersection(): test if a random rectangle intersection with itself
- testEquals(): Now checks for equality in the negative quadrant as well.
- testToString(): Check to see if negative quadrant integers are printing as expected.
- testContains(): Make sure a zero area rectangle is working as expected.

### 3. Negative Test: Christina Yu & Mario LoPrinzi: 2

- testInit(): the Rectangle's width and height should not be negative
- testArea(): Result is incorrect as the int value is overflowed

### 4. Fixed Incorrect Test Method: Christina Yu: 1

- testDiagonal()

### 5. Fixed Incorrect Code in Original Project: Mario LoPrinzi : 3

- Throw exception in Rectangle constructor for negative width and height
- Throw exception in Rectangle area for integer overflow
- Contains() returns false for rectangles that have no area to contain things in.

### 6. Screen Shots of Test Results:

- TestArea() (Positive & Negative test) & Throw an exception in area():

Finished after 0.022 seconds

Runs: 9/9

Errors: 0

Failures: 0

```
// Accessors for computed values
public int area() {
    // This checks for integer overflow from the multiplication.
    // Mario LoPrinzi
    if (width * height < 0) {
        throw new ArithmeticException("Width*Height has caused an integer overflow.");
    }
    return width * height;
}
```

```

@Test
public void testArea() {
    Rectangle z = new Rectangle(0, 0, 0, 0);
    // Testing the zero Rectangle
    assertEquals(z.area(), 0);

    Rectangle r = new Rectangle(0, 0, 2, 2);
    // Testing basic rectangle
    assertEquals(r.area(), 4);

    Rectangle s = new Rectangle(1, 2, 3, 4);
    // Testing away from origin
    assertEquals(s.area(), 12);

    // Testing the area of rectangle starting from random point with random width
    // and height, by Yu
    Random random = new Random();
    int a = random.nextInt(10000);
    int b = random.nextInt(10000);
    int c = Math.abs(random.nextInt(10000));
    int d = Math.abs(random.nextInt(10000));
    Rectangle t = new Rectangle(a, b, c, d);
    assertEquals(t.area(), c * d);

    // Testing the area value which is greater than the max value of int, by Yu
    Rectangle x = new Rectangle(a, b, (int) Math.pow(2, 32), (int) Math.pow(2, 32));
    // System.out.println("Expected Area of x is: "+ MAX_INT*MAX_INT);

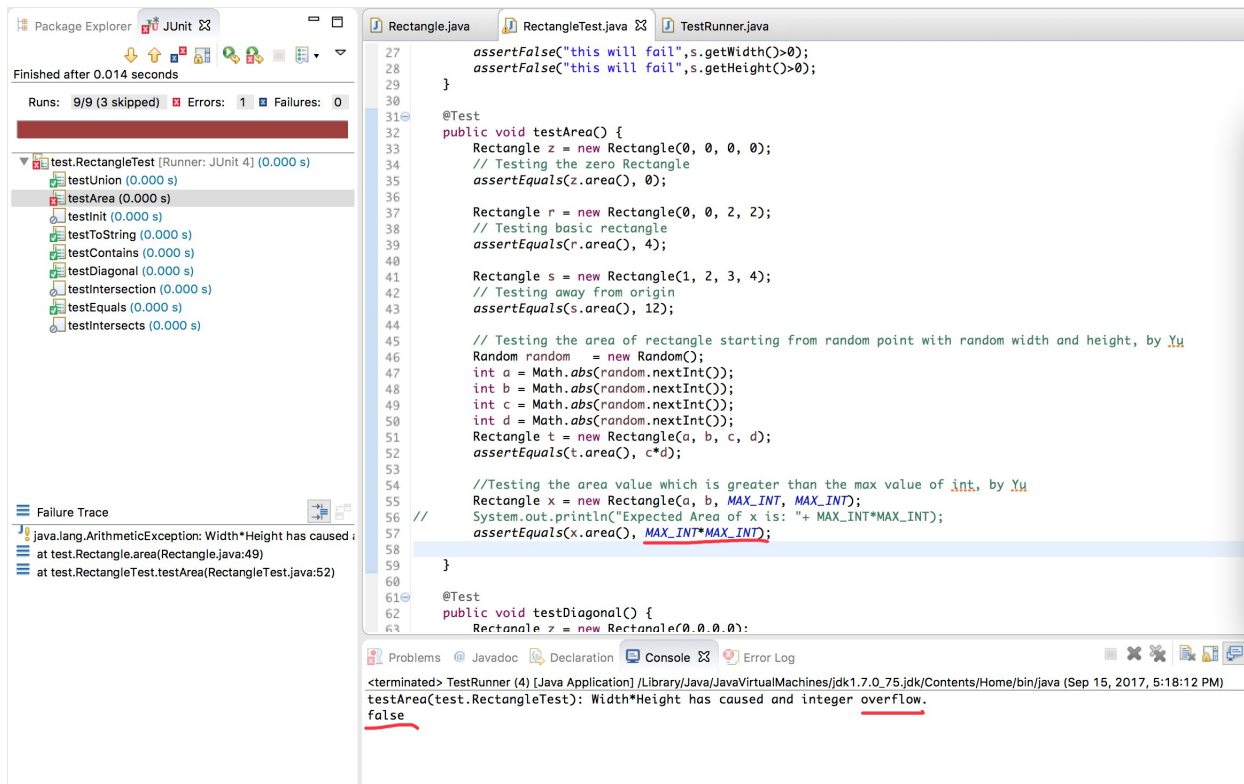
    // throws exception if area turns negative from integer overflow. Mario LoPrinzi
    try {
        x.area();
    } catch (ArithmeticException error) {
        assertEquals("Width*Height has caused an integer overflow.", error.getMessage());
    }
}

```

We generate random numbers to test rectangle area starting from random point with random width and height, as it may cover more situations.

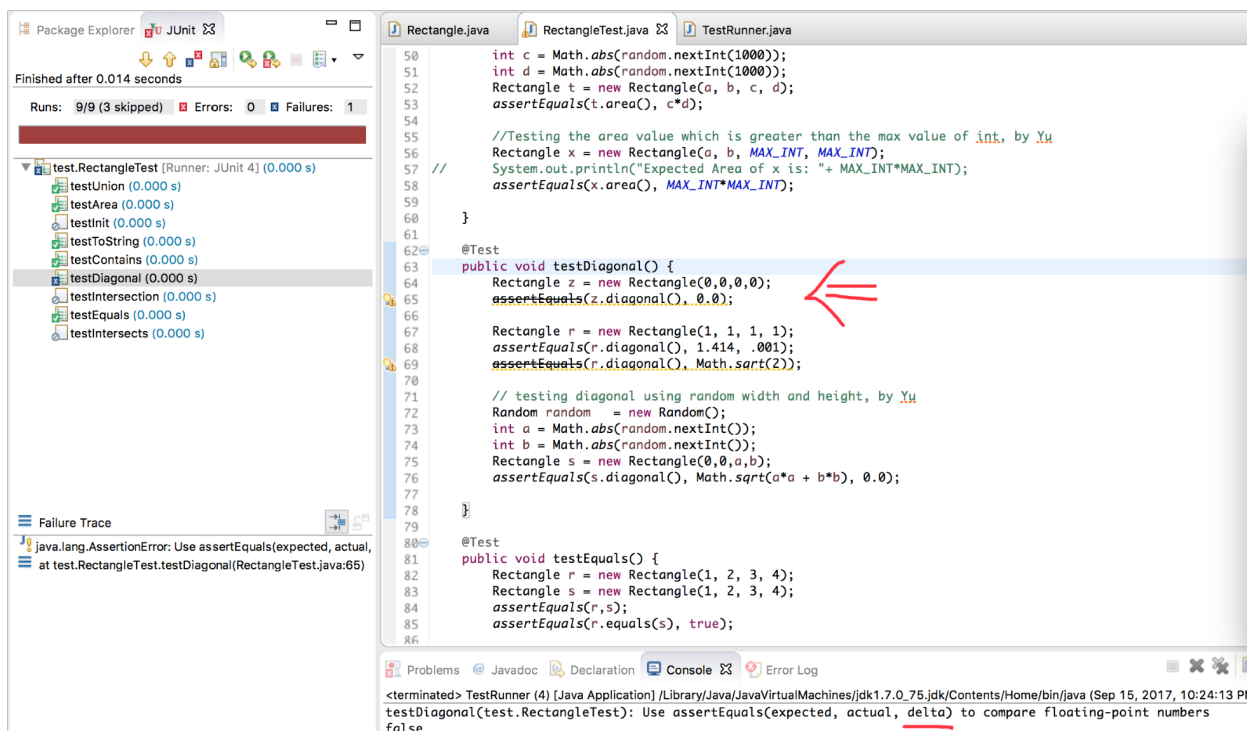
We wrote a negative test to check the area of the rectangle with the width and height both be `MAX_INT`— $2^{31}-1$ . Overflow happened, and the result of area become 1.

Then we threw an exception in `Rectangle area()` to catch the error above. This time we used the same negative test, and the program caught the overflow error.

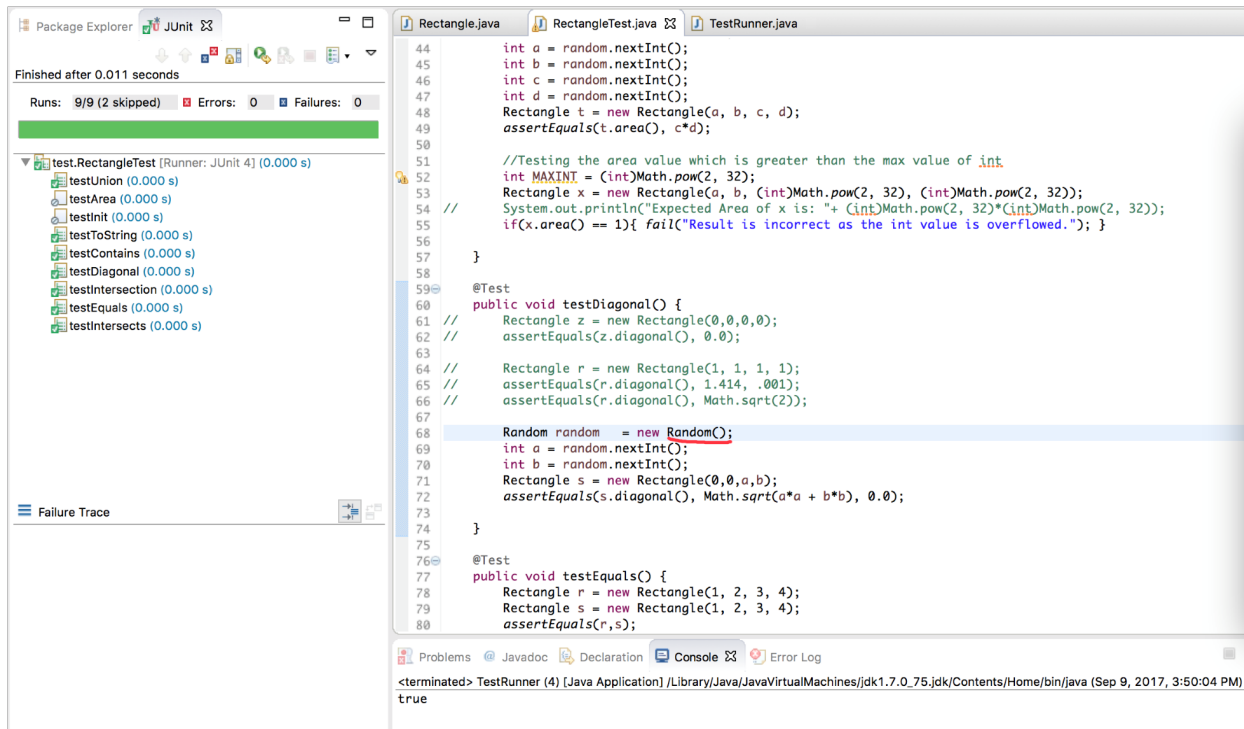


- TestDiagonal() & Fixed Incorrect Test Method

The TestDiagonal() code has an error with a parameter missing in the assertEquals(), and the test file could not be compiled. We fixed the error by adding the correct delta value.

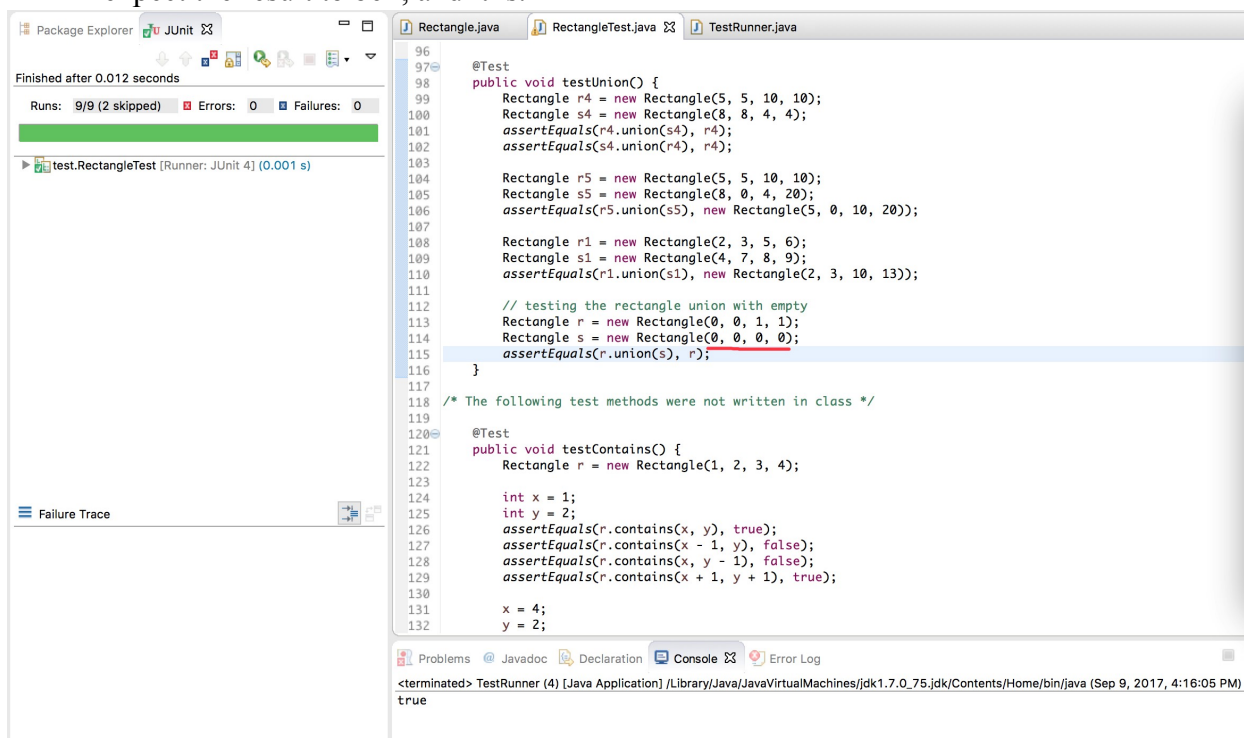


We wrote a positive test using the random generated numbers to test if the Diagonal() works, as it could cover more situations. The result passed.



- TestUnion()

We test a special situation as the rectangle r union with an empty rectangle(0, 0, 0, 0), we expect the result to be r, and it is.



- TestIntersects()

We test the situation when the given rectangle r intersects with a newly created rectangle of the same dimension as r, and the expected result should be r, and it is.

```

176 assertEquals(r.intersects(insideR), true);
177 assertEquals(insideR.intersects(r), true);
178
179 Rectangle s = new Rectangle(0, 5, 15, 5);
180 Rectangle noCornersInSButIntersects = new Rectangle(5, 0, 5, 15);
181 assertEquals(s.intersects(noCornersInSButIntersects), true);
182 assertEquals(noCornersInSButIntersects.intersects(s), true);
183
184 Rectangle t = new Rectangle(2, 0, 2, 1);
185 Rectangle horizontallyInsideTButNotIntersecting = new Rectangle(0, 5, 5, 3);
186 assertEquals(t.intersects(horizontallyInsideTButNotIntersecting), false);
187 assertEquals(horizontallyInsideTButNotIntersecting.intersects(t), false);
188
189 // modified by Yu
190 Rectangle topIntersectsR = new Rectangle(1, 1, 2, 5);
191 assertEquals(r.intersects(topIntersectsR), true);
192 // System.out.println(topIntersectsR.intersects(r));
193 assertEquals(topIntersectsR.intersects(r), true);
194 if(topIntersectsR.intersects(r) == false){
195     fail("Result should not be false");
196 }
197
198 // testing if rectangle intersects with itself by Yu
199 Rectangle x = r;
200 assertEquals(x.intersects(r), true);
201
202 }
203
204 @Test
205 public void testIntersection() {
206     Rectangle r = new Rectangle(0, 0, 4, 4);
207
208     Rectangle notIntersectR = new Rectangle(6, 6, 4, 4);
209     assertEquals(r.intersection(notIntersectR), null);
210     assertEquals(notIntersectR.intersection(r), null);
211
212     Rectangle ULIntersectsR = new Rectangle(2, 2, 3, 5);

```

- TestIntersection()

We test the situation when a randomly generated a rectangle intersection with itself, the expected result should still be itself, and it is. We ran the test 1000 times, they all passed.

```

228 assertEquals(topIntersectsR.intersection(r), new Rectangle(1, 1, 2, 3));
229
230 Rectangle leftIntersectsR = new Rectangle(1, 2, 4, 1);
231 assertEquals(r.intersection(leftIntersectsR), new Rectangle(1, 2, 3, 1));
232 assertEquals(leftIntersectsR.intersection(r), new Rectangle(1, 2, 3, 1));
233
234 Rectangle insideR = new Rectangle(1, 1, 2, 2);
235 assertEquals(r.intersection(insideR), insideR);
236 assertEquals(insideR.intersection(r), insideR);
237
238 Rectangle s = new Rectangle(0, 5, 15, 5);
239 Rectangle noCornersInSButIntersects = new Rectangle(5, 0, 5, 15);
240 assertEquals(s.intersection(noCornersInSButIntersects), new Rectangle(5, 5, 5, 5));
241 assertEquals(noCornersInSButIntersects.intersection(s), new Rectangle(5, 5, 5, 5));
242
243 Rectangle t = new Rectangle(2, 0, 2, 1);
244 Rectangle horizontallyInsideTButNotIntersecting = new Rectangle(0, 5, 5, 3);
245 assertEquals(t.intersection(horizontallyInsideTButNotIntersecting), null);
246 assertEquals(horizontallyInsideTButNotIntersecting.intersection(t), null);
247
248 // Test if a random rectangle intersection with itself, by Yu
249 Random random = new Random();
250 int i = 0;
251 for(i=0; i<1000; i++){
252     int a = Math.abs(random.nextInt(1000));
253     int b = Math.abs(random.nextInt(1000));
254     int c = Math.abs(random.nextInt(1000));
255     int d = Math.abs(random.nextInt(1000));
256     Rectangle x = new Rectangle(a, b, c, d);
257     Rectangle y = new Rectangle(a, b, c, d);
258     assertEquals(x.intersection(y), new Rectangle(a, b, c, d));
259 }
260 }
261
262 }
263
264

```



- TestInit() & Throw an exception in Rectangle constructor:

We wrote a negative test which contains 4 negative numbers representing the location and width and height. And the program passed as it should not have.

Finished after 0.018 seconds

Runs: 9/9    ❌ Errors: 0    🚩 Failures: 0

```
public class RectangleTest {

    // Made Rectangle throw an exception for negative parameters Mario LoPrinzi
    @Test(expected = InvalidParameterException.class)
    public void testInit() {
        Rectangle r = new Rectangle(1, 2, 3, 4);
        assertEquals(r.getX(), 1);
        assertEquals(r.getY(), 2);
        assertEquals(r.getWidth(), 3);
        assertEquals(r.getHeight(), 4);

        // testing with negative numbers, tested by Yu
        new Rectangle(-1, -1, -1, -1);
        // testing with negative width and height Mario LoPrinzi
        new Rectangle(0, 0, -1, -1);
    }
}
```

We threw an exception in the constructor to catch the error. Now the error message will be shown if the invalid value was input.

```
// Constructor declaration
Rectangle(int x, int y, int width, int height) {
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
    // Check for negative values so negative rectangles cannot be made.
    // Mario LoPrinzi
    if (this.width < 0 || this.height < 0) {
        throw new InvalidParameterException("Values must be positive.");
    }
}
```

- TestEquals()
 

We added a test for the lower left quadrant where the x and y of the Rectangle are negative.

```
@Test
public void testEquals() {
    Rectangle r = new Rectangle(1, 2, 3, 4);
    Rectangle s = new Rectangle(1, 2, 3, 4);
    assertEquals(r, s);
    assertEquals(r.equals(s), true);

    Rectangle t = new Rectangle(4, 3, 2, 1);
    assertEquals(r.equals(t), false);

    assertEquals(r.equals(new Object()), false);

    // Testing equality on the negative values. Mario LoPrinzi
    Rectangle lowerLeftR = new Rectangle(-2, -3, 4, 5);
    assertEquals(lowerLeftR.equals(new Rectangle(-2, -3, 4, 5)), true);
}
```

- TestToString()  
We added a negative test to make sure the test passes.

```

83 Rectangle r = new Rectangle(1, 2, 3, 4);
84 Rectangle s = new Rectangle(1, 2, 3, 4);
85 assertEquals(r, s);
86 assertEquals(r.equals(s), true);
87
88 Rectangle t = new Rectangle(4, 3, 2, 1);
89 assertEquals(r.equals(t), false);
90
91 assertEquals(r.equals(new Object()), false);
92
93 // Testing equality on the negative values. Mario LoPrinzi
94 Rectangle lowerLeftR = new Rectangle(-2, -3, 4, 5);
95 assertEquals(lowerLeftR.equals(new Rectangle(-2, -3, 4, 5)), true);
96
97
98 @Test
99 public void testToString() {
100     Rectangle r = new Rectangle(1, 2, 3, 4);
101     assertEquals("1,2, width = 3, height = 4", r.toString());
102     // Testing lower left quadrant rectangle. Mario LoPrinzi
103     Rectangle lowerLeftR = new Rectangle(-2, -3, 4, 5);
104     assertEquals("-2,-3, width = 4, height = 5", lowerLeftR.toString());
105 }
106
107 @Test
108 public void testUnion() {
109     Rectangle r4 = new Rectangle(5, 5, 10, 10);
110     Rectangle s4 = new Rectangle(8, 8, 4, 4);
111     assertEquals(r4.union(s4), r4);
112     assertEquals(s4.union(r4), r4);
113
114     Rectangle r5 = new Rectangle(5, 5, 10, 10);
115     Rectangle s5 = new Rectangle(8, 0, 4, 20);
116     assertEquals(r5.union(s5), new Rectangle(5, 0, 10, 20));
117
118     Rectangle r1 = new Rectangle(2, 3, 5, 6);
119     Rectangle s1 = new Rectangle(4, 7, 8, 9);
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

- TestContains()  
We test this to make sure the zero-area rectangle can not contain a point.

```

public boolean contains(int x, int y) {
    if(this.area() !=0){
        return this.x <= x && x <= this.x + width && this.y <= y && y <= this.y + height;
    }
    else
        return false;
}

```

```

149 y = 6;
150 assertEquals(r.contains(x, y), true);
151 assertEquals(r.contains(x - 1, y), false);
152 assertEquals(r.contains(x, y + 1), false);
153 assertEquals(r.contains(x + 1, y - 1), true);
154
155 x = 4;
156 y = 6;
157 assertEquals(r.contains(x, y), true);
158 assertEquals(r.contains(x + 1, y), false);
159 assertEquals(r.contains(x, y + 1), false);
160 assertEquals(r.contains(x - 1, y - 1), true);
161
162 // Testing a zero area Rectangle for expected behavior
163 // Mario LoPrinzi
164 x = 1;
165 y = 2;
166 r = new Rectangle(0, 0, 0, 0);
167 assertEquals(r.contains(x, y), false);
168 assertEquals(r.contains(x + 1, y), false);
169 assertEquals(r.contains(x, y + 1), false);
170 assertEquals(r.contains(x - 1, y - 1), false);
171 x = 0;
172 y = 0;
173 assertEquals(r.contains(x, y), false);
174 // Testing a zero area Rectangle for expected behavior
175 // Mario LoPrinzi
176 r = new Rectangle(0, 0, 0, 1);
177 assertEquals(r.contains(x, y), false);
178 assertEquals(r.contains(x + 1, y), false);
179 assertEquals(r.contains(x, y + 1), false);
180 assertEquals(r.contains(x - 1, y - 1), false);
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```