

Assertion Roulette

1. CheckoutTest

Projeto: Adyen/adyen-java-api-library

Arquivo: CheckoutTest.java

PR: <https://github.com/Adyen/adyen-java-api-library/pull/681>

Categorias que alteram: Assertion Roulette

Redução: 33

Aumento: 0

Motivo do Flaky: Comparações diretas entre strings formatadas em JSON podem ocasionalmente falhar em mudanças de ambiente, pois as ordens dos elementos em JSON não são preservadas.

Correção do Flaky: Para asserts que incluem comparação bruta de strings no formato JSON foi implementada uma função auxiliar `assertJsonStringEquals()`. A função faz comparações diretas após analisar strings de entrada com o método `JsonParser()` do Gson, permitindo a permutação de ordens de elementos em JSONs.

Motivo redução de Test Smells: A categoria de smells *Assertion Roulette* ocorre quando um teste contém mais de um assert sem mensagem explicativa. Como alguns testes foram refatorados para utilizar a função auxiliar `assertJsonStringEquals()`, a ferramenta TsDetect, detectou que a quantidade de *Assertion Roulette* diminuiu, porém continua a mesma. Segue o print da comparação antes e depois da correção do flaky.

Conclusão: Nesse caso a redução do test smells não influencia na correção do flaky test, isso porque a ferramenta não detectou o smells devido a chamada do assert em um novo método `assertJsonStringEquals()`, porém o teste continua fazendo o assert de maneira duplicada e sem mensagem explicativa, a diferença é que agora o assert está dentro de um método separado, fazendo com que a ferramenta TsDetect não detecte o smells. O desenvolvedor poderia fazer o parse do json dentro do próprio teste e chamar o *assertion statement*, o que também resolveria o flakiness, porém não reduziria o test smell.

```
558 @Test
559 public void TestSepaDirectDebitDetailsSerialization() throws JsonProcessingException {
560     String expectedJson = "{\"amount\":{\"value\":\"1000\",\"currency\":\"USD\"},\"merchantAccount\":\"MagentoMerchantTest\",\"paymentMethod\":{\"iban\":\"DE871234567891234567890\"}}";
561
562     SepaDirectDebitDetails sepaDirectDebitDetails = new SepaDirectDebitDetails();
563     sepaDirectDebitDetails.setOwnerName("A. Schneider");
564     sepaDirectDebitDetails.setIban("DE871234567891234567890");
565     PaymentsRequest paymentsRequest = createPaymentsCheckoutRequest();
566     paymentsRequest.setPaymentMethod(sepaDirectDebitDetails);
567
568     String gson = GSON.toJson(paymentsRequest);
569     assertEquals(expectedJson, gson);
570
571     String jackson = OBJECT_MAPPER.writeValueAsString(paymentsRequest);
572     assertEquals(expectedJson, jackson);
573 }
```

```

2400+
2401+     private void assertJsonStringEquals(String firstInput, String secondInput) {
2402+         JsonParser parser = new JsonParser();
2403+         assertEquals(parser.parse(firstInput), parser.parse(secondInput));
2404+     }

```

```

559 @Test
560 public void TestSepaDirectDebitDetailsSerialization() throws JsonProcessingException {
561     String expectedJson = "{\"amount\":{\"value\":1000,\"currency\":\"USD\"},\"merchantAccount\":\"MagentoMerchantTest\", \"paymentMethod\":{\"iban\":\"DE8712345678123456789\"}}";
562
563     SepaDirectDebitDetails sepaDirectDebitDetails = new SepaDirectDebitDetails();
564     sepaDirectDebitDetails.setOwnerName("A. Schneider");
565     sepaDirectDebitDetails.setIban("DE87123456781234567890");
566     PaymentsRequest paymentsRequest = createPaymentsCheckoutRequest();
567     paymentsRequest.setPaymentMethod(sepaDirectDebitDetails);
568
569     String gson = GSON.toJson(paymentsRequest);
570+    assertJsonStringEquals(expectedJson, gson);
571
572     String jackson = OBJECT_MAPPER.writeValueAsString(paymentsRequest);
573+    assertJsonStringEquals(expectedJson, jackson);
574

```

2. SaleToAcquirerDataSerializerTest

Projeto: Adyen/adyen-java-api-library

Arquivo: SaleToAcquirerDataSerializerTest.java

PR: <https://github.com/Adyen/adyen-java-api-library/pull/681>

Categorias que alteram: Assertion Roulette

Redução: 1

Aumento: 0

Motivo do Flaky: Comparações diretas entre strings formatadas em JSON podem ocasionalmente falhar em mudanças de ambiente, pois as ordens dos elementos em JSON não são preservadas.

Correção do Flaky: Para asserts que incluem comparação bruta de strings no formato JSON foi implementada uma função auxiliar `assertJsonStringEquals()`. A função faz comparações diretas após analisar strings de entrada com o método `JsonParser()` do Gson, permitindo a permutação de ordens de elementos em JSONs.

Motivo redução de Test Smells: A categoria de smells *Assertion Roulette* ocorre quando um teste contém mais de um assert sem mensagem explicativa. Como o teste foi refatorado para utilizar a função auxiliar `assertJsonStringEquals()`, a ferramenta TsDetect, detectou que a quantidade de *Assertion Roulette* diminuiu, porém continua a mesma, já que é utilizado mais de um assert para o mesmo teste, a diferença é que um dos assert foi refatorado para dentro da função auxiliar fazendo com que a ferramenta não detecte o smells.

Conclusão: Semelhante a análise anterior, a redução do test smells não influencia na correção do flaky test, isso porque a ferramenta não detectou o smells devido a chamada do assert em um novo método `assertJsonStringEquals()`, porém o teste continua fazendo o assert de maneira duplicada e sem mensagem explicativa, a diferença é que agora o assert está dentro de um método separado, fazendo com que a ferramenta TsDetect não detecte o smells. O desenvolvedor poderia fazer o parse do json dentro do próprio teste e chamar o *assertion statement*, o que também resolveria o flakiness, porém não reduziria o test smell.

```

100
101 // test if json string matches
102 String requestJson = PRETTY_PRINT_GSON.toJson(saleToAcquirerData);
103- assertEquals(requestJson, json);
104
105 // test if base64 works
106- String jsonBase64 = new String(Base64.encodeBase64(json.getBytes()));
107- assertEquals(jsonBase64, saleToAcquirerDataModelAdapter.serialize(saleToAcquirerData, null, null).getAsString());
108
109
110
111
112
113
114
115
116 }
117
118
119
120
121
122 // test if json string matches
123 String requestJson = PRETTY_PRINT_GSON.toJson(saleToAcquirerData);
124+ assertEquals(requestJson, json);
125
126 // test if base64 works
127+ String serialized = saleToAcquirerDataModelAdapter.serialize(saleToAcquirerData, null, null).getAsString();
128+ SaleToAcquirerData saleToAcquirerDataDecoded = new Gson().fromJson(new String(Base64.decodeBase64(serialized)), SaleToAcquirerData.class);
129+ assertEquals(saleToAcquirerData, saleToAcquirerDataDecoded);
130+
131+
132+
133+ public static void assertJsonStringEquals(String firstInput, String secondInput) {
134+     JsonParser parser = new JsonParser();
135+     assertEquals(parser.parse(firstInput), parser.parse(secondInput));
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

```

3. GsonFieldNamePolicyTest

Projeto: apache/camel-spring-boot

Arquivo: GsonFieldNamePolicyTest.java

PR: <https://github.com/apache/camel-spring-boot/pull/681>

Categorias que alteram: Assertion Roulette

Redução: 0

Aumento: 1

Motivo do Flaky: O método `template.requestBody()` que converte o objeto Pojo em String JSON não garante a ordem dos elementos, isso torna o resultado do teste não determinístico e o teste falha sempre que a função retorna uma incompatibilidade na ordem dos elementos na String JSON.

Correção do Flaky: No lugar de utilizar `assertEquals` para verificar o JSON como um todo, foi comparado cada elemento do JSON esperado utilizando `json.contains` tornando o teste mais determinístico e fazendo com que o flakiness seja removido

Motivo aumento de Test Smells: Foi adicionado um `assertTrue` para comparar cada elemento do JSON, como não foi adicionado nenhuma mensagem descritiva no `assertTrue` o teste sofreu aumento na categoria *Assertion Roulette*.

Conclusão: O aumento de test smells da categoria *Assertion Roulette* não influenciou na remoção do flakiness, a abordagem escolhida pelo desenvolvedor foi de comparar cada elemento do Json em um `assertTrue` separado o que removeu o flakiness mas aumentou a quantidade de test smells visto que nenhuma mensagem descritiva foi adicionada. Seria possível converter as string em Gson e comparar os objetos em um único `assert`, removendo o flakiness já que não dependeria da ordem dos elementos e não alterando a quantidade de test smells.

```

68  @Test
69  public void testMarshalPojo() {
70      PersonPojo pojo = new PersonPojo();
71      pojo.setId(123);
72      pojo.setFirstName("Donald");
73      pojo.setLastName("Duck");
74
75      String expected = "{\"id\":123,\"first_name\":\"Donald\",\"last_name\":\"Duck\"}";
76      String json = template.requestBody("direct:inPojo", pojo, String.class);
77      assertEquals(expected, json);
78  }

```

```

69  @Test
70  public void testMarshalPojo() {
71      PersonPojo pojo = new PersonPojo();
72      pojo.setId(123);
73      pojo.setFirstName("Donald");
74      pojo.setLastName("Duck");
75
76      String json = template.requestBody("direct:inPojo", pojo, String.class);
77+     assertTrue(json.contains("\"id\":123"));
78+     assertTrue(json.contains("\"first_name\":\"Donald\""));
79+     assertTrue(json.contains("\"last_name\":\"Duck\""));
80  }
81

```

4. GsonMarshalExclusionTest

Mesmo caso da análise do teste **GsonFieldNamePolicyTest** feita anteriormente, em relação a quantidade de redução, categoria, motivos e conclusão.

5. SpringGsonFieldNamePolicyTest

Mesmo caso da análise do teste **GsonFieldNamePolicyTest** feita anteriormente, em relação a quantidade de redução, categoria, motivos e conclusão.

6. TestGenerateJsonSchema

Projeto: FasterXML/jackson-databind

Arquivo: TestGenerateJsonSchema.java

PR: <https://github.com/FasterXML/jackson-databind/pull/3336>

Categorias que alteram: Assertion Roulette

Redução: 0

Aumento: 1

Motivo do Flaky: É realizada verificação de igualdade de um Json com uma string fixa, porém as propriedades do Json não possui uma ordem fixa ocasionando o flakiness.

Correção do Flaky: Foi utilizado ObjectNode/JsonNode para obter o valor de cada elemento do Json e comparar cada um individualmente removendo o flakiness.

Motivo aumento de Test Smells: A categoria *Assertion Roulette* ocorre quando em um teste possui mais de um *assertion statement* sem uma mensagem descritiva. A correção do flaky adicionou outros asserts no mesmo teste o que ocasionou o aumento de um smells da categoria *Assertion Roulette*.

Conclusão: O aumento de test smells da categoria *Assertion Roulette* não influenciou na remoção do flakiness, o desenvolvedor eliminou o problema comparando cada propriedade do json separadamente com um assertEquals o que ocasionou o aumento do test smells *Assertion*

Roulette, porém caso o desenvolvedor adicionasse mensagem descritiva nos asserts, ou mesmo desenvolvesse uma comparação entre os objetos não levando em consideração a ordem como no caso da comparação entre dois Gson, o flakiness seria removido e não teria alteração na quantidade de test smells.

```
229 // [databind#271]
230 public void testUnwrapping() throws Exception
231 {
232     JsonSchema jsonSchema = MAPPER.generateJsonSchema(UnwrappingRoot.class);
233     String json = jsonSchema.toString().replaceAll("\\\"", "");
234     String EXP = "{ 'type': 'object', "
235         + " 'properties': { 'age': { 'type': 'integer' }, "
236         + " 'name.first': { 'type': 'string' }, 'name.last': { 'type': 'string' } } }";
237     assertEquals(EXP, json);
238 }
```

```
229 // [databind#271]
230 public void testUnwrapping() throws Exception
231 {
232     JsonSchema jsonSchema = MAPPER.generateJsonSchema(UnwrappingRoot.class);
233+    ObjectNode root = jsonSchema.getSchemaNode();
234+    JsonNode propertiesSchema = root.get("properties");
235+    String ageType = propertiesSchema.get("age").get("type").asText();
236+    String firstType = propertiesSchema.get("name.first").get("type").asText();
237+    String lastType = propertiesSchema.get("name.last").get("type").asText();
238+    String type = root.get("type").asText();
239+    assertEquals(type, "object");
240+    assertEquals(ageType, "integer");
241+    assertEquals(firstType, "string");
242+    assertEquals(lastType, "string");
243 }
```

7. ValueReferenceTest

Projeto: Graylog2/graylog2-server

Arquivo: ValueReferenceTest.java

PR: <https://github.com/Graylog2/graylog2-server/pull/6908>

Categorias que alteram: Assertion Roulette

Redução: 5

Aumento: 0

Motivo do Flaky: Os testes usam jackson para serializar objetos em json e comparam com string fixas. No entanto, a ordem do json serializado não é garantida, portanto, os testes podem falhar se a ordem for diferente.

Correção do Flaky: É utilizado JSONAssert para comparação do Json pois a ordem dos elementos é desconsiderada na comparação e com isso o flakiness é removido.

Motivo redução de Test Smells: A categoria de smells *Assertion Roulette* ocorre quando um teste possui mais de um *assertion statement* sem mensagem descritiva. Como alguns testes foram refatorados para utilizar a função auxiliar `assertJsonEqualsNonStrict()`, os testes com

múltiplos asserts não foram detectados pela ferramenta TsDetect, já que o assert fica dentro da função auxiliar, com isso a quantidade de test smells foi reduzido, porém a quantidade de smells continua a mesma.

Conclusão: A redução do test smells não influencia na correção do flaky test, isso porque a ferramenta não detectou o smells devido a chamada do assert em um novo método `assertJsonStringEqualsNonStrict()`, porém o teste continua fazendo o assert de maneira duplicada e sem mensagem explicativa, a diferença é que agora o assert está dentro de um método separado, fazendo com que a ferramenta TsDetect não detecte o smells.

```
108 @Test
109 public void serializeBoolean() throws IOException {
110-     assertThat(objectMapper.writeValueAsString(ValueReference.of(true))).isEqualTo("{\"@type\":\"boolean\",\"@value\":true}");
111-     assertThat(objectMapper.writeValueAsString(ValueReference.of(false))).isEqualTo("{\"@type\":\"boolean\",\"@value\":false}");
112 }
113
114 @Test
115 public void deserializeBoolean() throws IOException {
116     assertThat(objectMapper.readValue("{\"@type\":\"boolean\",\"@value\":true}", ValueReference.class)).isEqualTo(ValueReference.of(true));
117     assertThat(objectMapper.readValue("{\"@type\":\"boolean\",\"@value\":false}", ValueReference.class)).isEqualTo(ValueReference.of(false));
118 }
119
120 @Test
121 public void serializeEnum() throws IOException {
122-     assertThat(objectMapper.writeValueAsString(ValueReference.of(TestEnum.A))).isEqualTo("{\"@type\":\"string\",\"@value\":\"A\"}");
123-     assertThat(objectMapper.writeValueAsString(ValueReference.of(TestEnum.B))).isEqualTo("{\"@type\":\"string\",\"@value\":\"B\"}");
124 }
```

```
36+ public void assertJsonEqualsNonStrict(String json1, String json2) {
37+     try {
38+         JSONAssert.assertEquals(json1, json2, false);
39+     } catch (JSONException jse) {
40+         throw new IllegalArgumentException(jse.getMessage());
41+     }
42+ }
```

```
118 @Test
119 public void serializeBoolean() throws IOException {
120+     assertJsonEqualsNonStrict(objectMapper.writeValueAsString(ValueReference.of(true)), json2:"{\"@type\":\"boolean\",\"@value\":true}");
121+     assertJsonEqualsNonStrict(objectMapper.writeValueAsString(ValueReference.of(false)), json2:"{\"@type\":\"boolean\",\"@value\":false}");
122 }
123
124 @Test
125 public void deserializeBoolean() throws IOException {
126     assertThat(objectMapper.readValue("{\"@type\":\"boolean\",\"@value\":true}", ValueReference.class)).isEqualTo(ValueReference.of(true));
127     assertThat(objectMapper.readValue("{\"@type\":\"boolean\",\"@value\":false}", ValueReference.class)).isEqualTo(ValueReference.of(false));
128 }
129
130 @Test
131 public void serializeEnum() throws IOException {
132+     assertJsonEqualsNonStrict(objectMapper.writeValueAsString(ValueReference.of(TestEnum.A)), json2:"{\"@type\":\"string\",\"@value\":\"A\"}");
133+     assertJsonEqualsNonStrict(objectMapper.writeValueAsString(ValueReference.of(TestEnum.B)), json2:"{\"@type\":\"string\",\"@value\":\"B\"}");
134 }
135 }
```

8. EnumUtilTest

Projeto: looly/hutool

Arquivo: EnumUtilTest.java

PR: <https://github.com/looly/hutool/pull/1270>

Categorias que alteram: Assertion Roulette,

Redução: 0

Aumento: 1

Motivo do Flaky: A lista de string retornada em `getFieldNames()` não é determinística e não possui uma ordem fixa causando o flakiness ao comparar com a lista gerada.

Correção do Flaky: Foi substituído o `assertEquals` por dois `assertTrue` onde é verificado se a lista de string possui os elementos “type” e “name” separadamente, o que torna o teste determinístico e remove o flakiness.

Motivo aumento de Test Smells: A categoria *Assertion Roulette* ocorre quando em um teste possui mais de um *assertion statement* sem uma mensagem descritiva. A correção do flaky

adicionou outros assert no teste getFieldNamesTest() o que ocasionou o aumento de um smells da categoria *Assertion Roulette*.

Conclusão: O aumento de test smells da categoria *Assertion Roulette* não influenciou na remoção do flakiness, o desenvolvedor removeu o flakiness ao verificar se a lista contém cada um dos elementos desejados, no lugar de comparar a lista inteira, com isso foi substituído um assertEquals por 2 assertTrue o que fez o test smells aumentar, já que não foi adicionado nenhuma mensagem descritiva. Seria possível corrigir o smells adicionando uma mensagem descritiva, assim teríamos a mesma quantidade de test smells porém sem a ocorrência do flakiness.

```
30  @Test
31  public void getFieldNamesTest() {
32      List<String> names = EnumUtil.getFieldNames(TestEnum.class);
33-  Assert.assertEquals(CollUtil.newArrayList("type", "name"), names);
34  }

30  @Test
31  public void getFieldNamesTest() {
32      List<String> names = EnumUtil.getFieldNames(TestEnum.class);
33+  Assert.assertTrue(names.contains(o:"type"));
34+  Assert.assertTrue(names.contains(o:"name"));
35  }
```

9. DirectedMultiGraphTest

Projeto: stanfordnlp/CoreNLP

Arquivo: DirectedMultiGraphTest.java

PR: <https://github.com/stanfordnlp/CoreNLP/pull/1215>

Categorias que alteram: Assertion Roulette

Redução: 1

Aumento: 0

Motivo do Flaky: O código original assume que graph.getConnectedComponents() retorna a lista de componentes em uma ordem específica. No entanto, este não é o caso, o que pode causar problemas durante o teste de porque o caso de teste pode falhar de forma não determinística.

Correção do Flaky: A correção essencialmente remove o conceito de ordem usando Sets. Isso funciona porque os componentes só precisam existir na mesma coleção; não é necessário que os componentes estejam em uma ordem específica.

Motivo redução de Test Smells: A categoria *Assertion Roulette* ocorre quando em um teste possui mais de um *assertion statement sem uma mensagem descritiva*. A correção do flaky removeu uma chamada de assertEquals, restando apenas uma no teste testConnectedComponents(), desta forma um smells foi reduzido da categoria *Assertion Roulette*.

Conclusão: A redução do test smells não tem relação com a correção do flakiness, nesse mesmo caso o desenvolvedor poderia ter apenas removido a comparação do tamanho da lista css presente na linha 160 do código com flakiness, ou adicionar uma mensagem descritiva, o que removeria o smells porém continuaria com o flakiness.

```

153 public void testConnectedComponents() {
154
155     System.out.println("graph is " + graph.toString());
156     List<Set<Integer>> ccs = graph.getConnectedComponents();
157     for (Set<Integer> cc : ccs) {
158         System.out.println("Connected component: " + cc);
159     }
160     assertEquals(ccs.size(), 4);
161     assertEquals(CollectionUtils.sorted(ccs.get(index:0)),
162         Arrays.asList(...a:1, 2, 3, 4));
163 }

```

```

153 public void testConnectedComponents() {
154
155     System.out.println("graph is " + graph.toString());
156+    Set<Set<Integer>> ccs = new HashSet<>(graph.getConnectedComponents());
157     for (Set<Integer> cc : ccs) {
158         System.out.println("Connected component: " + cc);
159     }
160+    Set<Integer> edge1 = new HashSet<>(Arrays.asList(...a:1, 2, 3, 4));
161+    Set<Integer> edge2 = new HashSet<>(Arrays.asList(...a:5, 6, 7));
162+    Set<Integer> edge3 = new HashSet<>(Arrays.asList(...a:8));
163+    Set<Integer> edge4 = new HashSet<>(Arrays.asList(...a:9,10));
164+    Set<Set<Integer>> expectedCcs = new HashSet<>(Arrays.asList(edge1,edge2,edge3,edge4));
165+    assertEquals(expectedCcs, ccs);
166 }

```

Conclusão da categoria Assertion Roulette

Foi observado que todos os casos de flakiness nessa categoria foi devido a comparação com strings fixas, o que torna o teste não determinístico caso a ordem de uma delas seja diferente. Todas as correções, embora com implementações diferentes, seguiram a ideia de comparar todas os casos possíveis para eliminar a dependência de ordem ou converter as strings em objetos e compará-los, a depender da abordagem escolhida a quantidade de test smells da categoria Assertion Roulette reduz ou aumenta, o que torna o argumento mais sólido de que as correções desse tipo de test smells não impacta na correção do flakiness presente no teste.

Sensitive Equality

1. Issue2430

Projeto: alibaba/fastjson

Arquivo: Issue2430.java

PR: <https://github.com/alibaba/fastjson/pull/3503>

Categorias que alteram: Sensitive Equality

Redução: 1

Aumento: 0

Motivo do Flaky: Os testes existentes são flaky (inconsistentes) porque dependem da ordem dos elementos em um Map.

Correção do Flaky: Para corrigir foi utilizado `SerializerFeature.MapSortField` para forçar a ordem ao converter JSON em string, para que a string convertida seja determinística.

Motivo redução de Test Smells: A categoria *Sensitive Equality* ocorre quando o método `toString` é usado em um teste. Com a utilização do método `toJSONString` junto com o

SerializerFeature.MapSortField para ordenar o map antes de converter em string, o método `toString` foi removido o que ocasionou a redução da categoria *Sensitive Equality*. Entretanto com esse ajuste o teste falha visto que está comparando o mesmo objeto com duas strings diferentes como mostrado na linha 22 e 28 da correção.

Conclusão: Nesse caso, a correção do test smell pode ocasionar a remoção do flakiness, pois a categoria *Sensitive Equality* ocorre aqui ao utilizar o método `toString` no `assertEquals`, para corrigir esse smells o desenvolvedor é obrigado a remover o método `toString` o que pode levar a comparação dos objetos no lugar da string fixa, removendo assim o test smells e consequentemente o flaky test.

```
9      public void testForIssue() {
10          ArrayListMultimap<String, String> multimap = ArrayListMultimap.create();
11          multimap.put("a", "1");
12          multimap.put("a", "2");
13          multimap.put("a", "3");
14          multimap.put("b", "1");
15
16          VO vo = new VO();
17          vo.setMap(multimap);
18          vo.setName("zhangsan");
19
20          assertEquals("{\"map\":{\"a\":[\"1\",\"2\",\"3\"],\"b\":[\"1\"]},\"name\":\"zhangsan\"}",
21+                      JSON.toJSONString(vo));
22      }
23
24      public void testForIssue2() {
25          String jsonString = "{\"map\":{\"a\":[\"1\",\"2\",\"3\"],\"b\":[\"1\"]},\"name\":\"zhangsan\"}";
26          VO vo = JSON.parseObject(jsonString, VO.class);
27+          assertEquals("VO:{name->zhangsan,map->{a=[1, 2, 3], b=[1]}}", vo.toString());
28      }
```

```
10      public void testForIssue() {
11          ArrayListMultimap<String, String> multimap = ArrayListMultimap.create();
12          multimap.put("a", "1");
13          multimap.put("a", "2");
14          multimap.put("a", "3");
15          multimap.put("b", "1");
16
17          VO vo = new VO();
18          vo.setMap(multimap);
19          vo.setName("zhangsan");
20
21          assertEquals("{\"map\":{\"a\":[\"1\",\"2\",\"3\"],\"b\":[\"1\"]},\"name\":\"zhangsan\"}",
22+                      JSON.toJSONString(vo, SerializerFeature.MapSortField));
23      }
24
25      public void testForIssue2() {
26          String jsonString = "{\"map\":{\"a\":[\"1\",\"2\",\"3\"],\"b\":[\"1\"]},\"name\":\"zhangsan\"}";
27          VO vo = JSON.parseObject(jsonString, VO.class);
28+          assertEquals("VO:{name->zhangsan,map->{a=[1, 2, 3], b=[1]}}", JSON.toJSONString(vo, SerializerFeature.MapSortField));
29      }
```

2. ResetVMUserDataCmdTest

Projeto: apache/cloudstack

Arquivo: ResetVMUserDataCmdTest.java

PR: <https://github.com/apache/cloudstack/pull/6967>

Categorias que alteram: Sensitive Equality

Redução: 1

Aumento: 0

Motivo do Flaky: O teste é flakiness pois compara a string de dois Hashmaps utilizando do método `toString()`, entretanto de acordo com a documentação, essa classe não oferece

garantias quanto à ordem do Map, ou seja, não garante que a ordem permanecerá constante ao longo do tempo.

Correção do Flaky: Em vez de converter os Hashmaps em string utilizando o método `toString()` e compará-los, o código foi refatorado para realizar a comparação entre os dois Hashmaps diretamente, o que é mais razoável e corrige o flakiness.

Motivo redução de Test Smells: A categoria *Sensitive Equality* ocorre quando o método `toString()` é usado em um teste. Com a correção para comparar os dois Hashmaps diretamente no lugar de converter em string, o método `toString()` foi removido o que ocasionou a redução da categoria *Sensitive Equality*.

Conclusão: A correção do smells impacta diretamente na correção do flakiness, pois a comparação entre duas string fixas é a causa da inconsistência, porém ao corrigir o smells *Sensitive Equality* o desenvolvedor é obrigado a remover o método `toString()`, no caso dessa refatoração foi realizado a comparação entre dois Hashmaps, o que não leva em consideração a ordem dos elementos, diferentemente da comparação entre as duas strings. Dessa forma, ao corrigir o test smells o desenvolvedor é forçado a resolver o flakiness do teste.

```
116 @Test
117 public void testUserdataDetails() {
118     Map<String, String> values1 = new HashMap<>();
119     values1.put("key1", "value1");
120     values1.put("key2", "value2");
121
122     Map<String, String> values2 = new HashMap<>();
123     values1.put("key3", "value3");
124     values1.put("key4", "value4");
125
126     Map<Integer, Map<String, String>> userdataDetails = new HashMap<>();
127     userdataDetails.put(0, values1);
128     userdataDetails.put(1, values2);
129
130     ReflectionTestUtils.setField(cmd, "userdataDetails", userdataDetails);
131
132     Map<String, String> result = cmd.getUserdataDetails();
133
134     values1.putAll(values2);
135-    Assert.assertEquals(values1.toString(), result.toString());
136 }
```

```
116 @Test
117 public void testUserdataDetails() {
118     Map<String, String> values1 = new HashMap<>();
119     values1.put("key1", "value1");
120     values1.put("key2", "value2");
121
122     Map<String, String> values2 = new HashMap<>();
123     values1.put("key3", "value3");
124     values1.put("key4", "value4");
125
126     Map<Integer, Map<String, String>> userdataDetails = new HashMap<>();
127     userdataDetails.put(0, values1);
128     userdataDetails.put(1, values2);
129
130     ReflectionTestUtils.setField(cmd, "userdataDetails", userdataDetails);
131
132     Map<String, String> result = cmd.getUserdataDetails();
133
134     values1.putAll(values2);
135+    Assert.assertEquals(values1, result);
136 }
```

3. JsonIncludePropertiesTest

Projeto: FasterXML/jackson-annotations

Arquivo: JsonIncludePropertiesTest.java

PR: <https://github.com/FasterXML/jackson-annotations/pull/194>

Categorias que alteram: Sensitive Equality

Redução: 1

Aumento: 0

Motivo do Flaky: No teste `testFromAnnotation()`, a variável `included` é um set. Portanto, a ordem de 'foo' e 'bar' não é fixa. Ao verificar a igualdade o teste pode falhar ocasionalmente.

Correção do Flaky: Foi adicionado um assert que compara o objeto com todas as variações na ordem de 'foo' e 'bar', assim independentemente da ordem dos elementos o teste irá passar, removendo o flakiness.

Motivo redução de Test Smells: A categoria *Sensitive Equality* ocorre quando um teste utiliza o método `toString()` em um assertion statement. A correção do flaky removeu a utilização do método `toString()` de dentro do assert e atribuiu a variável `tmp` ocasionando a não detecção do smells pela ferramenta e diminuindo um smell da categoria *Sensitive Equality*.

Conclusão: Nesse caso, a correção do test smell pode ocasionar a remoção do flakiness, como o desenvolvedor é obrigado a remover o `toString` do *assertion statement*, a refatoração tende a remover a comparação com strings fixas, nesse teste foi comparado todas as possibilidades da string. Nota-se que o método `toString` ainda é utilizado como visto na linha 39 da correção, porém por não estar dentro do *assertion statement* o TsDetect não identificou o test smells, uma melhoria é necessária no TsDetect para identificar esses casos, mas mesmo com isso a correção desse tipo de smells é capaz de remover o flakiness do código, pois força o desenvolvedor a utilizar outros tipo de comparação no lugar de string fixas.

```
30 public void testFromAnnotation()
31 {
32     JsonIncludeProperties.Value v = JsonIncludeProperties.Value.from(Bogus.class.getAnnotation(annotationClass:JsonIncludeProperties.class));
33     assertNotNull(v);
34     Set<String> included = v.getIncluded();
35     assertEquals(2, v.getIncluded().size());
36     assertEquals(_set(...args:"foo", "bar"), included);
37     assertEquals("JsonIncludeProperties.Value(included=[bar, foo])", v.toString());
38
39     assertEquals(v, JsonIncludeProperties.Value.from(Bogus.class.getAnnotation(annotationClass:JsonIncludeProperties.class)));
40 }

32 public void testFromAnnotation()
33 {
34     JsonIncludeProperties.Value v = JsonIncludeProperties.Value.from(Bogus.class.getAnnotation(annotationClass:JsonIncludeProperties.class));
35     assertNotNull(v);
36     Set<String> included = v.getIncluded();
37     assertEquals(2, v.getIncluded().size());
38     assertEquals(_set(...args:"foo", "bar"), included);
39+    String tmp = v.toString();
40+    boolean test1 = tmp.equals(anObject:"JsonIncludeProperties.Value(included=[foo, bar])");
41+    boolean test2 = tmp.equals(anObject:"JsonIncludeProperties.Value(included=[bar, foo])");
42+    assertTrue(test1 || test2);
43     assertEquals(v, JsonIncludeProperties.Value.from(Bogus.class.getAnnotation(annotationClass:JsonIncludeProperties.class)));
44 }
```

4. BinaryInTextTest

Projeto: OpenHFT/Chronicle-Wire

Arquivo: BinaryInTextTest.java

PR: <https://github.com/OpenHFT/Chronicle-Wire/pull/310>

Categorias que alteram: Sensitive Equality

Redução: 1

Aumento: 0

Motivo do Flaky: Em alguns momentos o teste `testReserialize()` falha pois é realizado uma comparação com string fixa, entretanto a ordem do elementos em `bit.toString()` não é garantida.

Correção do Flaky: Foi considerado as 2 possíveis sequências na comparação, o que removeu o flakiness.

Motivo redução de Test Smells: A categoria *Sensitive Equality* ocorre quando um teste utiliza o método `toString()` em um assertion statement. A correção do flaky removeu a utilização do

método `toString()` de dentro do `assert` e atribuiu a variável `bitToString` ocasionando a não detecção do smells pela ferramenta.

Conclusão: Semelhante a conclusão anterior, a correção do test smell pode ocasionar a remoção do flakiness pois o desenvolvedor é obrigado a remover o `toString` do *assertion statement*, forçando-o a utilizar outros tipo de comparação no lugar de string fixas, nesse caso não foi utilizado comparação de objeto para remover a inconsistência, mas sim comparado todas as combinações possíveis da string. Pelo código percebemos que o método `toString` ainda permanece porém fora do `assert`, uma melhoria seria necessária no `TsDetect` para considerar esses casos, porém mesmo assim conclui-se que a correção do test smells impacta diretamente na remoção do flakiness.

```
26 @Test
27 public void testReserialize() {
28     BIT b = new BIT();
29     byte[] a = new byte[5];
30     b.b = Bytes.wrapForRead(a);
31     b.c = Bytes.wrapForRead(a);
32     // System.out.println(b);
33
34     BIT bit = Marshallable.fromString(BIT.class, "{\n" +
35         "b: !!binary AAAAAA=\n" +
36         "c: !!binary CCCCCC\n" +
37         "}");
38     assertEquals("!inet.openhft.chronicle.wire.BinaryInTextTest$BIT {\n" +
39         "  b: !!binary AAAAAA=\n" +
40         "  c: !!binary CCCCCC\n" +
41         "}" + "\n", bit.toString());
42 }
```

```
27 @Test
28 public void testReserialize() {
29     BIT b = new BIT();
30     byte[] a = new byte[5];
31     b.b = Bytes.wrapForRead(a);
32     b.c = Bytes.wrapForRead(a);
33     // System.out.println(b);
34
35     BIT bit = Marshallable.fromString(BIT.class, "{\n" +
36         "b: !!binary AAAAAA=\n" +
37         "c: !!binary CCCCCC\n" +
38         "}");
39     String bitToString = bit.toString();
40     assertTrue(bitToString.equals("!inet.openhft.chronicle.wire.BinaryInTextTest$BIT {\n" +
41         "  b: !!binary AAAAAA=\n" +
42         "  c: !!binary CCCCCC\n" +
43         "}" + "\n") ||
44         bitToString.equals("!inet.openhft.chronicle.wire.BinaryInTextTest$BIT {\n" +
45         "  c: !!binary CCCCCC\n" +
46         "  b: !!binary AAAAAA=\n" +
47         "}" + "\n"));
48 }
```

5. LoggingIfValidationFailsTest

Projeto: rest-assured/rest-assured

Arquivo: LoggingIfValidationFailsTest.java

PR: <https://github.com/rest-assured/rest-assured/pull/1280>

Categorias que alteram: Sensitive Equality

Redução: 1

Aumento: 0

Motivo do Flaky: A ordem do JSON não é garantida, portanto, os testes podem passar ou falhar a depender da ordem dos parâmetros do Json.

Correção do Flaky: A correção feita foi utilizar `JSONAssert` para verificar os resultados do JSON de maneira mais segura. Então o comportamento não determinístico foi eliminado tornando o teste estável.

Motivo redução de Test Smells: A categoria *Sensitive Equality* ocorre quando um teste utiliza o método `toString()` em um `assertion statement`. A correção do flaky removeu a utilização do método `toString()` de dentro do `assert` e atribui a variável `writerString` ocasionando a não

detecção do smells pela ferramenta TsDetect, portanto a redução não foi concreta, apenas a ferramenta não detectou.

Conclusão: A correção do test smell nessa situação também pode remover o flakiness do código, pois como é realizada a comparação entre duas string fixas para corrigir o test smell é necessário remover o método toString do assert, o que leva a comparação ser realizada de outras maneiras, como nesse caso a utilização do JsonAssert para comparar os objetos Json que desconsidera a ordem dos elementos, diferentemente do toString onde a ordem dos elementos é relevante.

```
71- } catch (AssertionError e) {
72-     assertThat(writer.toString(), equalTo(String.format("Request method:\tPOST\n" +
73-         "Request URI:\thttp://localhost:8080/greetingPost\n" +
74-         "Proxy:\t\t\t\t\t\n" +
75-         "Request params:\tname=Johan\n" +
76-         "Query params:\t\t\t\t\t\n" +
77-         "Form params:\t\t\t\t\t\n" +
78-         "Path params:\t\t\t\t\t\n" +
79-         "Headers:\t\tContent-Type=application/x-www-form-urlencoded;charset=%s\n" +
80-         "Cookies:\t\t\t\t\t\n" +
81-         "Multiparts:\t\t\t\t\t\n" +
82-         "\nBody:\t\t\t\t\t\n" +
83-         "\n" +
84-         "200\n" +
85-         "Content-Type: application/json;charset=UTF-8\n" +
86-         "\n" +
87-         "{\n  \"id\": 1,\n  \"content\": \"Hello, Johan!\n\n\"}"));
88-     RestAssuredMockMvcConfig.config().getEncoderConfig().defaultContentCharset());
89- }
90- }
```

```
58+ public void
59+ assertJsonEqualsNonStrict(String json1, String json2) {
60+     try {
61+         JSONAssert.assertEquals(json1, json2, false);
62+     } catch (JSONException jse) {
63+         throw new IllegalArgumentException(jse.getMessage());
64+     }
65+ }
66+ }
```

```
83- } catch (AssertionError e) {
84-     String writerString = writer.toString();
85-     String headerString = String.format("Request method:\tPOST\n" +
86-         "Request URI:\thttp://localhost:8080/greetingPost\n" +
87-         "Proxy:\t\t\t\t\t\n" +
88-         "Request params:\tname=Johan\n" +
89-         "Query params:\t\t\t\t\t\n" +
90-         "Form params:\t\t\t\t\t\n" +
91-         "Path params:\t\t\t\t\t\n" +
92-         "Headers:\t\tContent-Type=application/x-www-form-urlencoded;charset=%s\n" +
93-         "Cookies:\t\t\t\t\t\n" +
94-         "Multiparts:\t\t\t\t\t\n" +
95-         "\nBody:\t\t\t\t\t\n" +
96-         "\n" +
97-         "200\n" +
98-         "Content-Type: application/json;charset=UTF-8\n" +
99-         RestAssuredMockMvcConfig.config().getEncoderConfig().defaultContentCharset());
100-     assertThat(writerString, startsWith(headerString));
101-     assertJsonEqualsNonStrict(writerString.replace(headerString, replacement:"").trim(),
102-         json2:"{\n  \"id\": 1,\n  \"content\": \"Hello, Johan!\n\n\"}");
103- }
104- }
```

6. ResponseLoggingTest

Mesmo caso da análise do teste **LoggingIfValidationFailsTest** feita anteriormente, em relação a quantidade de redução, categoria, motivos e conclusão.

7. CollectionValuedMapTest

Projeto: stanfordnlp/CoreNLP

Arquivo: CollectionValuedMapTest.java

PR: <https://github.com/stanfordnlp/CoreNLP/pull/1182>

Categorias que alteram: Sensitive Equality

Redução: 1

Aumento: 0

Motivo do Flaky: O teste testAddRemove() assume que há uma ordem específica na interação do HashMap. No entanto, a documentação do java não especifica uma ordem, tornando o teste não determinístico.

Correção do Flaky: A correção feita foi realizar a comparação entre duas `CollectionValuedMap` no lugar de verificar as strings, assim a ordem dos elementos é irrelevante removendo o flakiness do teste.

Motivo redução de Test Smells: A categoria *Sensitive Equality* ocorre quando um teste utiliza o método `toString()` em um assertion statement. A correção do flaky removeu a comparação com uma string fixa, não necessitando utilizar o método `toString()` em um `CollectionValuedMap` diminuindo um smells da categoria *Sensitive Equality*.

Conclusão: A correção do test smells impacta diretamente na correção do flakiness, pois a comparação entre duas string fixas é a causa da inconsistência, porém ao corrigir o smells *Sensitive Equality* o desenvolvedor é obrigado a remover o método `toString`, no caso dessa refatoração foi realizado a comparação entre dois Maps que não leva em consideração a ordem dos elementos, diferentemente da comparação entre as duas strings. Dessa forma, ao corrigir o test smells o desenvolvedor é forçado a resolver o flakiness do teste.

```
184  @Test
185  public void testAddRemove() {
186      CollectionValuedMap<Integer, Integer> fooMap = new CollectionValuedMap<>();
187
188      for (int i = 0; i < 4; i++) {
189          for (int j = 0; j < 4; j++) {
190              fooMap.add(new Integer(i), new Integer(j));
191          }
192      }
193      fooMap.remove(new Integer(value:2));
194      Assert.assertEquals("{0=[0, 1, 2, 3], 1=[0, 1, 2, 3], 3=[0, 1, 2, 3]}", fooMap.toString());
195  }
```

```
187  @Test
188  public void testAddRemove() {
189      CollectionValuedMap<Integer, Integer> fooMap = new CollectionValuedMap<>();
190      CollectionValuedMap<Integer, Integer> expectedMap = new CollectionValuedMap<>();
191      for (int i = 0; i < 4; i++) {
192          for (int j = 0; j < 4; j++) {
193              fooMap.add(new Integer(i), new Integer(j));
194              if (i!=2){
195                  expectedMap.add(new Integer(i), new Integer(j));
196              }
197          }
198      }
199      fooMap.remove(new Integer(value:2));
200      Assert.assertEquals(expectedMap,fooMap);
201  }
```

8. JSONPointerTest

Projeto: stleary/JSON-java

Arquivo: JSONPointerTest.java

PR: <https://github.com/stleary/JSON-java/pull/696>

Categorias que alteram: Sensitive Equality

Redução: 1

Aumento: 0

Motivo do Flaky: O teste falha ao comparar uma string esperada e o resultado da função `org.json.junit.JSONPointerTest.query()` após convertê-la em String. A função `toString` da classe `Object` não garante a ordem dos atributos no objeto. Isso torna o resultado do teste não determinístico e o teste falha sempre que o `toString` altera a ordem das propriedades.

Correção do Flaky: Os valores esperados e reais foram convertidos em `JSONObject` e utilizado `similar()` os objetos. Como esta função compara os valores dentro dos `JSONObjects` sem precisar de ordem, o teste se torna determinístico e garante que a flakiness do teste seja removida.

Motivo redução de Test Smells: A categoria *Sensitive Equality* ocorre quando um teste utiliza o método `toString()` em um `assertion statement`. A correção do flaky removeu o método `toString()` e passou a comparar dois `JsonObjects`, o esperado e o real, reduzindo um smell da categoria *Sensitive Equality*.

Conclusão: A comparação entre duas strings fixas devido a utilização do método `toString` é removido ao corrigir o test smells *Sensitive Equality*, o que remove também o flakiness que é causado devido essa comparação. Como foi refatorado para verificar a igualdade entre dois objetos, a dependência da ordem dos elementos do objeto foi removida, eliminando o comportamento inconsistente. Com isso conclui-se que a correção do test smells também elimina o flakiness desse teste.

```
73  @Test
74  public void queryByEmptyKeySubObject() {
75-    assertEquals( "{\\\":\\\"empty key of an object with an empty key\\\",\\\"subKey\\\":\\\"Some\" +
76-      \" other value\\\"}", query(pointer: "/obj/").toString());
77  }
```

```
73  @Test
74  public void queryByEmptyKeySubObject() {
75+    JSONObject json = new JSONObject("{\"\\\":\\\"empty key of an object with an empty key\\\",\\\"subKey\\\":\\\"Some\" +
76+      \" other value\\\"}");
77+    JSONObject obj = (JSONObject) query(pointer: "/obj/");
78+    assertTrue(json.similar(obj));
79  }
```

9. TestMetadata

Projeto: apache/tika

Arquivo: TestMetadata.java

PR: <https://github.com/apache/tika/pull/845>

Categorias que alteram: Sensitive Equality

Redução: 1

Aumento: 0

Motivo do Flaky: A classe `Metadata` armazena a informação em um `HashMap`, o que não garante a ordem dos elementos. Portanto a ordem dos elementos podem ser alteradas ao utilizar a função `toString()`. Como é realizado comparação direta entre strings, e a ordem não pode ser garantida, o teste se torna não determinístico.

Correção do Flaky: A comparação entre strings foi removida, para isso foi verificado se o `map` contém cada elemento individualmente, o que desconsidera a ordenação dos elementos tornando o teste determinístico e removendo o flakiness.

Motivo redução de Test Smells: A categoria *Sensitive Equality* ocorre quando um teste utiliza o método `toString()` em um `assertion statement`. A correção do flaky removeu a utilização do método `toString()` de dentro do `assert` e atribuiu a variável `metadata` ocasionando a não detecção do smells pela ferramenta e diminuindo um smell da categoria *Sensitive Equality*.

Conclusão: Nesse caso, a correção do test smell pode ocasionar a remoção do flakiness, como o desenvolvedor é obrigado a remover o `toString` do `assert`, a refatoração tende a remover a comparação com strings fixas, nesse teste foi verificado se cada elemento está presente no `HashMap` individualmente. Nota-se que o método `toString` ainda é utilizado como visto na linha 482 da correção, porém por não estar dentro do *assertion statement* o `TsDetect` não identificou o test smells, uma melhoria é necessária no `TsDetect` para identificar esses

casos, porém a correção desse tipo de smells é capaz de remover o flakiness do código, pois obriga o desenvolvedor a utilizar outro tipo de comparação no lugar de string fixas.

```
475     @Test
476     public void testToStringWithManyEntries() {
477         Metadata m = new Metadata();
478         m.add("key", "value1");
479         m.add("key", "value2");
480         m.add("key2", "value12");
481-        assertEquals("key2=value12 key=value1 key=value2", m.toString());
482    }

476     @Test
477     public void testToStringWithManyEntries() {
478         Metadata m = new Metadata();
479         m.add("key", "value1");
480         m.add("key", "value2");
481         m.add("key2", "value12");
482+        String metadata = m.toString();
483+        assertContains("key=value1", metadata);
484+        assertContains("key=value2", metadata);
485+        assertContains("key2=value12", metadata);
486    }
```

Conclusão da categoria *Sensitive Equality*

Foi observado que em todos os testes dessa categoria a inconsistência é devido a comparação entre strings fixas, que leva em consideração a ordem dos elementos. Além disso, a correção dos smells força o desenvolvedor a remover o método toString do assert, fazendo-o desenvolver outra solução no lugar de comparar as strings, o que faz com que o flakiness seja removido do teste.

Unknown Test

1. FastJsonpHttpMessageConverter4Case1Test

Projeto: alibaba/fastjson

Arquivo: FastJsonpHttpMessageConverter4Case1Test.java

PR: <https://github.com/alibaba/fastjson/pull/3569>

Categorias que alteram: Unknown Test

Redução: 2

Aumento: 0

Motivo do Flaky: Os testes existentes são flaky (inconsistentes) porque dependem da ordem dos elementos em um objeto JSON.

Correção do Flaky: Para corrigir foi adicionado manualmente as possíveis ordenações dos elementos do JSON na comparação

Motivo redução de Test Smells: A categoria *Unknown Test* ocorre quando o teste não contém uma única instrução de assert e o parâmetro de anotação `@Test(expect)`, dessa forma ao corrigir o Flaky foi adicionado o método `assertTrue` em 2 testes para comparar as possíveis

combinações de elementos do Json com o valor recebido o que ocasionou a redução de test smells para essa categoria.

Conclusão: A redução do test smells não altera o flakiness do código, o assert pode ser adicionado ao código sem necessariamente verificar todas as combinações para o Json o que reduziria a quantidade de smells porém a inconsistência permaneceria.

```
90 @Test
91 public void test2_2() throws Exception {
92
93     String jsonStr = "[{"name":"p1","sonList":[{"name":"s1"}]},{name":"p2","sonList":[{"name":"s2"}, {"name":"s3"}]}]";
94
95     ResultActions actions = mockMvc.perform(post("/fastjson/test2?jsonp=fnUpdateSome").characterEncoding("UTF-8")
96         .content(jsonStr).contentType(MediaType.APPLICATION_JSON));
97     actions.andDo(print());
98     actions.andExpect(status().isOk()).andExpect(content().contentType(APPLICATION_JAVASCRIPT))
99     .andExpect(content().string("/**/fnUpdateSome({"p1":1,"p2":2})"));
100 }
```

```
93 @Test
94 public void test2_2() throws Exception {
95
96     String jsonStr = "[{"name":"p1","sonList":[{"name":"s1"}]},{name":"p2","sonList":[{"name":"s2"}, {"name":"s3"}]}]";
97
98     ResultActions actions = mockMvc.perform(post("/fastjson/test2?jsonp=fnUpdateSome").characterEncoding("UTF-8")
99         .content(jsonStr).contentType(MediaType.APPLICATION_JSON));
100     actions.andDo(print());
101+    actions.andExpect(status().isOk()).andExpect(content().contentType(APPLICATION_JAVASCRIPT));
102+    String content = actions.andReturn().getResponse().getContentAsString();
103+    assertTrue(content.equals("/**/fnUpdateSome({"p1":1,"p2":2})"));
104+    || content.equals("/**/fnUpdateSome({"p2":2,"p1":1})"));
105 }
```

2. FastJsonpHttpMessageConverter4Case2Test

Mesmo caso da análise do teste **FastJsonpHttpMessageConverter4Case1Test** feita anteriormente, em relação a quantidade de redução, categoria, motivos e conclusão.

3. FastJsonpHttpMessageConverter4Case3Test

Mesmo caso da análise do teste **FastJsonpHttpMessageConverter4Case1Test** feita anteriormente, em relação a quantidade de redução, categoria, motivos e conclusão.

Conclusão da categoria Unknown Test

Foi observado que os casos de flakiness nessa categoria foi devido a dependência na ordem dos elementos do Json, o que torna o teste não determinístico, pois a ordem pode ser diferente. As correções adicionaram *assertion statement* no código o que reduziu a quantidade de test smells dessa categoria, porém essa redução não foi responsável por alterar ou influenciar a ocorrência de flakiness no código.

Exception Catching Throwing

1. IndexLabelTest

Projeto: apache/incubator-hugegraph-toolchain

Arquivo: IndexLabelTest.java

PR: <https://github.com/apache/incubator-hugegraph-toolchain/pull/398>

Categorias que alteram: Exception Catching Throwing

Redução: 0

Aumento: 3

Motivo do Flaky: Devido à ordem não determinística das propriedades na string JSON criada pelo método `JsonUtil.toJson()`. Ao realizar a comparação com uma string fixa pode acontecer dos elementos do JSON não estarem na ordem esperada.

Correção do Flaky: Foi utilizado o jackson `ObjectMapper` para comparar os elementos do `Json` em vez de strings brutas, o que torna o teste determinístico, pois a ordem não será comparada.

Motivo aumento de Test Smells: A categoria *Exception Catching Throwing* ocorre quando um teste contém uma instrução `throw` ou uma cláusula `catch`. Os desenvolvedores devem utilizar o tratamento de exceção do JUnit para passar/reprovar automaticamente no teste. Como na correção do flaky foi adicionado em cada um dos 3 testes o lançamento da exceção `JsonProcessingException` a categoria *Exception Catching Throwing* sofreu um aumento de 3 smells.

Conclusão: Verificar conclusão geral da categoria.  Flaky Analysis - Conclusão Thalisson

```
32  @Test
33- public void testIndexLabel() {
34      IndexLabel.Builder builder = new IndexLabel.BuilderImpl("personByAge",
35                                                              null);
36      IndexLabel indexLabel = builder.onV("person")
37                                  .secondary()
38                                  .by("age")
39                                  .build();
40
41      String json = "{ \"name\": \"personByAge\", \"id\": 0, \"
42                    \"check_exist\": true, \"user_data\": {}, \"
43                    \"base_type\": \"VERTEX_LABEL\", \"
44                    \"base_value\": \"person\", \"
45                    \"index_type\": \"SECONDARY\", \"fields\": [\"age\"], \"
46                    \"rebuild\": true }";
47-     Assert.assertEquals(json, JsonUtil.toJson(indexLabel));
48
49     Assert.assertEquals(HugeType.INDEX_LABEL.string(), indexLabel.type());
50 }
```

```
34  @Test
35+ public void testIndexLabel() throws JsonProcessingException {
36      IndexLabel.Builder builder = new IndexLabel.BuilderImpl("personByAge",
37                                                              null);
38      IndexLabel indexLabel = builder.onV("person")
39                                  .secondary()
40                                  .by("age")
41                                  .build();
42
43      String json = "{ \"name\": \"personByAge\", \"id\": 0, \"
44                    \"check_exist\": true, \"user_data\": {}, \"
45                    \"base_type\": \"VERTEX_LABEL\", \"
46                    \"base_value\": \"person\", \"
47                    \"index_type\": \"SECONDARY\", \"fields\": [\"age\"], \"
48                    \"rebuild\": true }";
49+     ObjectMapper mapper = new ObjectMapper();
50+     Assert.assertEquals(mapper.readTree(json), mapper.readTree(JsonUtil.toJson(indexLabel)));
51     Assert.assertEquals(HugeType.INDEX_LABEL.string(), indexLabel.type());
52 }
```

2. AvroSchemaTest

Projeto: apache/pulsar

Arquivo: AvroSchemaTest.java

PR: <https://github.com/apache/pulsar/pull/6247>

Categorias que alteram: Exception Catching Throwing

Redução: 0

Aumento: 2

Motivo do Flaky: O método `avroSchema.getSchemaInfo().getSchema()` retorna um objeto Json que é comparado com uma string fixa, entretanto o método não garante a ordem dos elementos do Json, tornando o teste flaky.

Correção do Flaky: Foi utilizado `JSONAssert` para verificar a igualdade entre Json, já que o `JSONAssert` desconsidera a ordem dos elementos no Json, tornando o teste determinístico e estável.

Motivo aumento de Test Smells: A categoria *Exception Catching Throwing* ocorre quando um teste contém uma instrução `throw` ou uma cláusula `catch`. Os desenvolvedores devem utilizar o tratamento de exceção do JUnit para passar/reprovar automaticamente no teste. Como na correção do flaky foi adicionado em 2 testes o lançamento da exceção `JSONException` a categoria *Exception Catching Throwing* sofreu um aumento de 2 smells.

Conclusão: Verificar conclusão geral da categoria.  Flaky Analysis - Conclusão Thalisson

```
145 @Test
146 public void testNotAllowNullSchema() {
147     AvroSchema<Foo> avroSchema = AvroSchema.of(SchemaDefinition.<Foo>builder().withPojo(Foo.class).withAlwaysAllowNull(false).build());
148     assertEquals(avroSchema.getSchemaInfo().getType(), SchemaType.AVRO);
149     Schema.Parser parser = new Schema.Parser();
150     String schemaJson = new String(avroSchema.getSchemaInfo().getSchema());
151     assertEquals(schemaJson, SCHEMA_AVRO_NOT_ALLOW_NULL);
152     Schema schema = parser.parse(schemaJson);
153
154     for (String fieldName : FOO_FIELDS) {
155         Schema.Field field = schema.getField(fieldName);
156         Assert.assertNotNull(field);
157
158         if (field.name().equals("field4")) {
159             Assert.assertNotNull(field.schema().getTypes().get(1).getField("field1"));
160         }
161         if (field.name().equals("fieldUnableNull")) {
162             Assert.assertNotNull(field.schema().getType());
163         }
164     }
165 }

146+ public void assertEquals(String s1, String s2) throws JSONException {
147+     JSONAssert.assertEquals(s1, s2, false);
148+ }
149+
150
151 @Test
152+ public void testNotAllowNullSchema() throws JSONException {
153     AvroSchema<Foo> avroSchema = AvroSchema.of(SchemaDefinition.<Foo>builder().withPojo(Foo.class).withAlwaysAllowNull(false).build());
154     assertEquals(avroSchema.getSchemaInfo().getType(), SchemaType.AVRO);
155     Schema.Parser parser = new Schema.Parser();
156     String schemaJson = new String(avroSchema.getSchemaInfo().getSchema());
157+     assertEquals(schemaJson, SCHEMA_AVRO_NOT_ALLOW_NULL);
158     Schema schema = parser.parse(schemaJson);
159
160     for (String fieldName : FOO_FIELDS) {
161         Schema.Field field = schema.getField(fieldName);
162         Assert.assertNotNull(field);
163
164         if (field.name().equals("field4")) {
165             Assert.assertNotNull(field.schema().getTypes().get(1).getField("field1"));
166         }
167         if (field.name().equals("fieldUnableNull")) {
168             Assert.assertNotNull(field.schema().getType());
169         }
170     }
171 }
```

3. JSONSchemaTest

Projeto: apache/pulsar

Arquivo: JSONSchemaTest_17f71d3.java

PR: <https://github.com/apache/pulsar/pull/6435>

Categorias que alteram: Exception Catching Throwing

Redução: 0

Aumento: 2

Motivo do Flaky: Semelhante ao que acontece no teste `AvroSchemaTest`, método `jsonSchema.getSchemaInfo().getSchema()` retorna um objeto Json que é comparado com uma

string fixa, entretanto o método não garante a ordem dos elementos do Json, tornando o teste flaky.

Correção do Flaky: Foi utilizado JSONAssert para verificar a igualdade entre Json, já que o JSONAssert desconsidera a ordem dos elementos no Json, tornando o teste determinístico e estável.

Motivo aumento de Test Smells: A categoria *Exception Catching Throwing* ocorre quando um teste contém uma instrução throw ou uma cláusula catch. Como na correção do flaky foi adicionado em 2 testes o lançamento da exceção JSONException a categoria *Exception Catching Throwing* sofreu um aumento de 2 smells.

Conclusão: Verificar conclusão geral da categoria.  Flaky Analysis - Conclusão Thalisson

```
47 @Test
48 public void testNotAllowNullSchema() {
49     JSONSchema<Foo> jsonSchema = JSONSchema.of(SchemaDefinition.<Foo>builder().withPojo(Foo.class).withAlwaysAllowNull(false).build());
50     Assert.assertEquals(jsonSchema.getSchemaInfo().getType(), SchemaType.JSON);
51     Schema.Parser parser = new Schema.Parser();
52     String schemaJson = new String(jsonSchema.getSchemaInfo().getSchema());
53     Assert.assertEquals(schemaJson, SCHEMA_JSON_NOT_ALLOW_NULL);
54     Schema schema = parser.parse(schemaJson);
55
56     for (String fieldName : FOO_FIELDS) {
57         Schema.Field field = schema.getField(fieldName);
58         Assert.assertNotNull(field);
59
60         if (field.name().equals("field4")) {
61             Assert.assertNotNull(field.schema().getTypes().get(1).getField("field1"));
62         }
63         if (field.name().equals("fieldUnableNull")) {
64             Assert.assertNotNull(field.schema().getType());
65         }
66     }
67 }

49+ public static void assertJSONEqual(String s1, String s2) throws JSONException{
50+     JSONAssert.assertEquals(s1, s2, false);
51+ }
52 @Test
53+ public void testNotAllowNullSchema() throws JSONException {
54     JSONSchema<Foo> jsonSchema = JSONSchema.of(SchemaDefinition.<Foo>builder().withPojo(Foo.class).withAlwaysAllowNull(false).build());
55     Assert.assertEquals(jsonSchema.getSchemaInfo().getType(), SchemaType.JSON);
56     Schema.Parser parser = new Schema.Parser();
57     String schemaJson = new String(jsonSchema.getSchemaInfo().getSchema());
58+     assertJSONEqual(schemaJson, SCHEMA_JSON_NOT_ALLOW_NULL);
59     Schema schema = parser.parse(schemaJson);
60
61     for (String fieldName : FOO_FIELDS) {
62         Schema.Field field = schema.getField(fieldName);
63         Assert.assertNotNull(field);
64
65         if (field.name().equals("field4")) {
66             Assert.assertNotNull(field.schema().getTypes().get(1).getField("field1"));
67         }
68         if (field.name().equals("fieldUnableNull")) {
69             Assert.assertNotNull(field.schema().getType());
70         }
71     }
72 }
```

4. JobRegistryTest

Projeto: apache/shardingsphere-elasticjob

Arquivo: JobRegistryTest.java

PR: <https://github.com/apache/shardingsphere-elasticjob/pull/625>

Categorias que alteram: Exception Catching Throwing

Redução: 0

Aumento: 1

Motivo do Flaky: Caso o teste assertGetCurrentShardingTotalCountIfNull() seja executado após o teste assertGetCurrentShardingTotalCountIfNotNull() ele irá falhar pois o teste não reseta o estado após sua execução.

Correção do Flaky: O estado foi resetado no final do teste

assertGetCurrentShardingTotalCountIfNotNull() para garantir que o teste

assertGetCurrentShardingTotalCountIfNull() não falhe caso seja executado posteriormente

Motivo aumento de Test Smells: A categoria *Exception Catching Throwing* ocorre quando um teste contém uma instrução throw ou uma cláusula catch. Na correção do flaky foi adicionado o lançamento da exceção `NoSuchFieldException` em 1 teste, com isso a categoria *Exception Catching Throwing* aumentou em 1 a quantidade de smells.

Conclusão: Verificar conclusão geral da categoria.  Flaky Analysis - Conclusão Thalisson

```
66  @Test
67  public void assertGetCurrentShardingTotalCountIfNull() {
68      assertThat(JobRegistry.getInstance().getCurrentShardingTotalCount("exist_job_instance"), is(0));
69  }
70
71  @Test
72- public void assertGetCurrentShardingTotalCountIfNotNull() {
73      JobRegistry.getInstance().setCurrentShardingTotalCount("exist_job_instance", 10);
74      assertThat(JobRegistry.getInstance().getCurrentShardingTotalCount("exist_job_instance"), is(10));
75  }
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
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
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
```

```

63  @Test
64- public void test1() {
65      String metadata = testFeign.test();
66-      Assertions.assertThat(metadata).isEqualTo("{\"a\":\"11\",\"b\":\"22\",\"c\":\"33\"}{\"LOCAL_SERVICE\":\"test\"
67-      + "\",\"LOCAL_PATH\":\"/test\",\"LOCAL_NAMESPACE\":\"default\"}");
68-
69      Assertions.assertThat(metadataLocalProperties.getContent().get("a")).isEqualTo("1");
70      Assertions.assertThat(metadataLocalProperties.getContent().get("b")).isEqualTo("2");
71      Assertions.assertThat(MetadataContextHolder.get().getTransitiveCustomMetadata("a")).isEqualTo("11");
72      Assertions.assertThat(MetadataContextHolder.get().getTransitiveCustomMetadata("b")).isEqualTo("22");
73      Assertions.assertThat(MetadataContextHolder.get().getTransitiveCustomMetadata("c")).isEqualTo("33");
74  }
75
76  @Test
77+ public void test1() throws JsonProcessingException {
78+     String metadata = testFeign.test();
79+     ObjectMapper mapper = new ObjectMapper();
80+     Assertions.assertThat(mapper.readTree(metadata)).isEqualTo(mapper.readTree("{\"a\":\"11\",\"b\":\"22\",\"c\":\"33\"}{\"LOCAL_SERVICE\":\"test\"
81+     + "\",\"LOCAL_PATH\":\"/test\",\"LOCAL_NAMESPACE\":\"default\"}"));
82+
83+     Assertions.assertThat(metadataLocalProperties.getContent().get("a")).isEqualTo("1");
84+     Assertions.assertThat(metadataLocalProperties.getContent().get("b")).isEqualTo("2");
85+     Assertions.assertThat(MetadataContextHolder.get().getTransitiveCustomMetadata("a")).isEqualTo("11");
86+     Assertions.assertThat(MetadataContextHolder.get().getTransitiveCustomMetadata("b")).isEqualTo("22");
87+     Assertions.assertThat(MetadataContextHolder.get().getTransitiveCustomMetadata("c")).isEqualTo("33");
88+ }

```

Conclusão da categoria Exception Catching Throwing

Todos os testes tiveram apenas aumento na quantidade de smells, isso ocorreu pois a correção do flakiness foi adicionada aos testes a anotação de lançamento de exceção, sendo detectado pelo TsDetect como um smells. Entretanto esse aumento não tem relação com o flakiness presente no código, e portanto conclui-se que a remoção do smells dessa categoria não irá alterar o comportamento flakiness do teste

Conditional Test Logic

1. ReflectionTest

Projeto: pholser/junit-quickcheck

Arquivo: ReflectionTest.java

PR: <https://github.com/pholser/junit-quickcheck/pull/283>

Categorias que alteram: Conditional Test Logic

Redução: 0

Aumento: 1

Motivo do Flaky: A ordem em que as anotações Z e W são obtidas no método allAnnotations não é garantida o que torna o teste flaky.

Correção do Flaky: Adicionado uma condicional para verificar qual anotação vem no index 2, se é a Z ou W para posteriormente verificar a anotação do index 3.

Motivo aumento de Test Smells: A categoria *Conditional Test Logic* ocorre quando o teste possui condicional, o que pode levar a situações em que o teste falha em detectar defeitos no método de produção, pois as instruções de teste não foram executadas porque uma condição não foi atendida. Após a correção do Flaky foi adicionado uma condicional para identificar qual assertEquals executar a depender da condicional, com isso a quantidade de smells sofreu um aumento nessa categoria.

Conclusão: O aumento do test smells não impacta na remoção do flakiness, durante a refatoração o desenvolvedor utilizou de más práticas, o que fez aumentar a quantidade de

smells. Esse cenário será desconsiderado para a análise.

```
545 @Test public void findingAnnotationsRecursively() {
546     Method method =
547         findMethod(this.getClass(), "withMarker", String.class);
548
549     List<Annotation> annotations =
550         allAnnotations(method.getParameters()[0]);
551
552     assertEquals(4, annotations.size());
553     assertEquals(X.class, annotations.get(index:0).annotationType());
554     assertEquals(Y.class, annotations.get(index:1).annotationType());
555     assertEquals(Z.class, annotations.get(index:2).annotationType());
556
557     assertEquals(W.class, annotations.get(index:3).annotationType());
558 }
559 }
```

```
545 @Test public void findingAnnotationsRecursively() {
546     Method method =
547         findMethod(this.getClass(), "withMarker", String.class);
548
549     List<Annotation> annotations =
550         allAnnotations(method.getParameters()[0]);
551
552     assertEquals(4, annotations.size());
553     assertEquals(X.class, annotations.get(index:0).annotationType());
554     assertEquals(Y.class, annotations.get(index:1).annotationType());
555     assertTrue(Z.class.equals(annotations.get(index:2).annotationType()) ||
556         W.class.equals(annotations.get(index:2).annotationType()));
557     if (Z.class.equals(annotations.get(index:2).annotationType())) {
558         assertEquals(W.class, annotations.get(index:3).annotationType());
559     } else {
560         assertEquals(Z.class, annotations.get(index:3).annotationType());
561     }
562 }
```

Conclusão da categoria Conditional Test Logic

Quantidade de testes insuficientes para uma conclusão.

Multiple Categories

Assertion Roulette e Sensitive Equality

1. ParseDistSQLHandlerTest

Projeto: apache/shardingsphere

Arquivo: ParseDistSQLHandlerTest.java

PR: <https://github.com/apache/shardingsphere/pull/22526>

Categorias que alteram: Assertion Roulette, Sensitive Equality

Redução: 1

Aumento: 2

Motivo do Flaky: O flakiness ocorre porque os objetos JSON estão sendo comparados e a ordem dos elementos no JSON gerado pode não ser determinística.

Correção do Flaky: Foi utilizado o gson `JsonParser.parseString()` para converter as strings em Json parse trees, dessa forma as parse trees são comparadas ignorando a ordem dos elementos, o que torna o teste determinístico e remove o flakiness.

Motivo redução de Test Smells: A categoria *Assertion Roulette* ocorre quando em um teste possui mais de um *assertion statement* sem uma mensagem descritiva. A correção do flaky removeu em um teste um dos dois `assertThat`, restando apenas um no teste e ocasionando a redução na quantidade de smells da categoria *Assertion Roulette*.

Motivo aumento de Test Smells: A categoria *Sensitive Equality* ocorre quando um teste utiliza o método `toString()`. Na correção do flaky foi utilizado o método `toString()` antes de converter em uma parse tree usando o método `JsonParser.parseString()`, como essa mudança foi realizada em 2 testes, a categoria *Sensitive Equality* teve um aumento de 2 smells.

Conclusão: A redução do test smells não influencia na correção do flakiness para esse teste, o desenvolvedor removeu um `assert`, restando apenas um `assert` no teste `assertGetRowDataForPostgreSQL()` o que ocasionou a redução, porém o desenvolvedor poderia ter realizado apenas essa alteração, o que faria a quantidade de smells reduzir, porém o flakiness não seria corrigido.

```
68 @Test
69 public void assertGetRowDataForMySQL() throws SQLException {
70     String sql = "select * from t_order";
71     when(connectionSession.getProtocolType()).thenReturn(new MySQLDatabaseType());
72     ParseStatement parseStatement = new ParseStatement(sql);
73     ParseDistSQLHandler parseDistSQLHandler = new ParseDistSQLHandler();
74     parseDistSQLHandler.init(parseStatement, connectionSession);
75     parseDistSQLHandler.execute();
76     parseDistSQLHandler.next();
77     SQLStatement statement = sqlParserRule.getSQLParserEngine("MySQL").parse(sql, false);
78     assertThat(new LinkedList<>(parseDistSQLHandler.getRowData().getData()).getFirst(), is("MySQLSelectStatement"));
79     assertThat(new LinkedList<>(parseDistSQLHandler.getRowData().getData()).getLast(), is(new Gson().toJson(statement)));
80 }
81
82 @Test
83 public void assertGetRowDataForPostgreSQL() throws SQLException {
84     String sql = "select * from t_order";
85     when(connectionSession.getProtocolType()).thenReturn(new PostgreSQLDatabaseType());
86     ParseStatement parseStatement = new ParseStatement(sql);
87     ParseDistSQLHandler parseDistSQLHandler = new ParseDistSQLHandler();
88     parseDistSQLHandler.init(parseStatement, connectionSession);
89     parseDistSQLHandler.execute();
90     parseDistSQLHandler.next();
91     SQLStatement statement = sqlParserRule.getSQLParserEngine("PostgreSQL").parse(sql, false);
92     assertThat(new LinkedList<>(parseDistSQLHandler.getRowData().getData()).getFirst(), is("PostgreSQLSelectStatement"));
93     assertThat(new LinkedList<>(parseDistSQLHandler.getRowData().getData()).getLast(), is(new Gson().toJson(statement)));
94 }
95
```

```
69 @Test
70 public void assertGetRowDataForMySQL() throws SQLException {
71     String sql = "select * from t_order";
72     when(connectionSession.getProtocolType()).thenReturn(new MySQLDatabaseType());
73     ParseStatement parseStatement = new ParseStatement(sql);
74     ParseDistSQLHandler parseDistSQLHandler = new ParseDistSQLHandler();
75     parseDistSQLHandler.init(parseStatement, connectionSession);
76+    parseDistSQLHandler.execute();
77     parseDistSQLHandler.next();
78     SQLStatement statement = sqlParserRule.getSQLParserEngine("MySQL").parse(sql, false);
79     assertThat(new LinkedList<>(parseDistSQLHandler.getRowData().getData()).getFirst(), is("MySQLSelectStatement"));
80+    assertThat(JsonParser.parseString(new LinkedList<>(parseDistSQLHandler.getRowData().getData()).getLast().toString()), is(JsonParser.parseString(new Gson().toJson(statement))));
81 }
82
83 @Test
84 public void assertGetRowDataForPostgreSQL() throws SQLException {
85     String sql = "select * from t_order";
86     when(connectionSession.getProtocolType()).thenReturn(new PostgreSQLDatabaseType());
87     ParseStatement parseStatement = new ParseStatement(sql);
88     ParseDistSQLHandler parseDistSQLHandler = new ParseDistSQLHandler();
89     parseDistSQLHandler.init(parseStatement, connectionSession);
90     parseDistSQLHandler.execute();
91     parseDistSQLHandler.next();
92     SQLStatement statement = sqlParserRule.getSQLParserEngine("PostgreSQL").parse(sql, false);
93+    assertThat(JsonParser.parseString(new LinkedList<>(parseDistSQLHandler.getRowData().getData()).getLast().toString()), is(JsonParser.parseString(new Gson().toJson(statement))));
94 }
95
```

2. UnmodifiableMultiValuedMapTest

Projeto: apache/commons-collections

Arquivo: UnmodifiableMultiValuedMapTest.java

PR: <https://github.com/apache/commons-collections/pull/190>

Categorias que alteram: Assertion Roulette, Sensitive Equality

Redução: 6

Aumento: 0

Motivo do Flaky: Várias funções no arquivo utiliza `assertEquals` entre uma string fixa e um `map.toString()`, onde o `map` é da classe `MultiValuedMap`, o problema ocorre pois o método `toString()` retorna uma string com elementos ordenados arbitrariamente.

No caso do `map` `{one=[uno, un], two=[dos, deux], three=[tres, trois]}`, o método `toString()` pode retornar `"{ two=[dos, deux], one=[uno, un], three=[tres, trois]}"`.

Correção do Flaky: Foi adicionado uma função auxiliar (helper function) no qual compara cada elemento do map individualmente tornando o teste determinístico e evitando que a ordem do map interfira no resultado do teste a cada execução. Com isso os asserts afetados foram substituídos pela função auxiliar.

Motivo redução de Test Smells: A categoria *Sensitive Equality* ocorre quando o método `toString` é usado em um teste, 3 testes foram refatorados para utilizar a função auxiliar `assertMapContainsAllValues()` no qual a comparação é realizada dentro da função, com isso o método `toString()` foi removido dos testes e passou a ser utilizado dentro do assert da função auxiliar ocasionando uma redução de 3 smells da categoria *Sensitive Equality* devido a não detecção dos smells pela ferramenta TsDetect, porém o smells continua acontecendo.

A categoria de smells *Assertion Roulette* ocorre quando um teste possui mais de um *assert* sem mensagem explicativa. Antes da correção do flaky, os 3 testes que sofreram alteração possuíam o `assertEquals()` e o `fail()` ocasionando o smells *Assertion Roulette* já que eles estão sem mensagem. Após a correção, o `assertEquals()` dos 3 testes foram substituídos pela função auxiliar `assertMapContainsAllValues` deixando apenas o `fail()` como *assertion statement* em cada teste, fazendo com que a ferramenta não detecte os smells e reduzindo 3 smells da categoria *Assertion Roulette*, entretanto os smells ainda estão presentes pois os `assert` continuam sendo chamados nos testes, porém dentro da função auxiliar.

Conclusão: A redução do test smells da categoria *Sensitive Equality*, impacta diretamente na remoção do flakiness do código pois obriga o desenvolvedor a remover o método `toString` do `assert`, nesse caso o flakiness ocorreu devido a comparação entre strings fixas, sendo que o método `toString()` do map não garante a ordem dos elementos. Entretanto, a solução feita pelo desenvolvedor de comparar cada elemento do Map ainda utiliza o método `toString` mas não foi detectado pela ferramenta devido o `assert` está sendo executado em uma função auxiliar. A categoria *Assertion Roulette* não altera o flakiness do código pois a ferramenta parou de detectar devido a remoção do `assertEquals` de dentro do teste para um função auxiliar, e no lugar de executar o `assert`, é feito a chamada da função.

```

104 public void testRemoveException() {
105     final MultiValuedMap<K, V> map = makeFullMap();
106     try {
107         map.remove("one");
108         fail();
109     } catch (final UnsupportedOperationException e) {
110         // expected, not support remove() method
111         // UnmodifiableMultiValuedMap does not support change
112     }
113     assertEquals("{one=[uno, un], two=[dos, deux], three=[tres, trois]}", map.toString());
114 }
115
116 public void testRemoveMappingException() {
117     final MultiValuedMap<K, V> map = makeFullMap();
118     try {
119         map.removeMapping("one", "uno");
120         fail();
121     } catch (final UnsupportedOperationException e) {
122         // expected, not support removeMapping() method
123         // UnmodifiableMultiValuedMap does not support change
124     }
125     assertEquals("{one=[uno, un], two=[dos, deux], three=[tres, trois]}", map.toString());
126 }
127
128 public void testClearException() {
129     final MultiValuedMap<K, V> map = makeFullMap();
130     try {
131         map.clear();
132         fail();
133     } catch (final UnsupportedOperationException e) {
134         // expected, not support clear() method
135         // UnmodifiableMultiValuedMap does not support change
136     }
137     assertEquals("{one=[uno, un], two=[dos, deux], three=[tres, trois]}", map.toString());
138 }

```

```

55+ private void assertMapContainsAllValues(MultiValuedMap<K, V> map) {
56+     assertEquals("[uno, un]", map.get((K) "one").toString());
57+     assertEquals("[dos, deux]", map.get((K) "two").toString());
58+     assertEquals("[tres, trois]", map.get((K) "three").toString());
59+ }

```

```

114 public void testRemoveException() {
115     final MultiValuedMap<K, V> map = makeFullMap();
116     try {
117         map.remove("one");
118         fail();
119     } catch (final UnsupportedOperationException e) {
120         // expected, not support remove() method
121         // UnmodifiableMultiValuedMap does not support change
122     }
123+     this.assertMapContainsAllValues(map);
124 }
125
126 public void testRemoveMappingException() {
127     final MultiValuedMap<K, V> map = makeFullMap();
128     try {
129         map.removeMapping("one", "uno");
130         fail();
131     } catch (final UnsupportedOperationException e) {
132         // expected, not support removeMapping() method
133         // UnmodifiableMultiValuedMap does not support change
134     }
135+     this.assertMapContainsAllValues(map);
136 }
137
138 public void testClearException() {
139     final MultiValuedMap<K, V> map = makeFullMap();
140     try {
141         map.clear();
142         fail();
143     } catch (final UnsupportedOperationException e) {
144         // expected, not support clear() method
145         // UnmodifiableMultiValuedMap does not support change
146     }
147+     this.assertMapContainsAllValues(map);
148 }

```

Assertion Roulette e Exception Catching Throwing

3. KeyValueSchemaInfoTest

Projeto: apache/pulsar

Arquivo: KeyValueSchemaInfoTest_17f71d3.java

PR: <https://github.com/apache/pulsar/pull/6435>

Categorias que alteram: Assertion Roulette, Exception Catching Throwing

Redução: 1

Aumento: 1

Motivo do Flaky: Semelhante aos casos anteriores, a variável *havePrimitiveType* é comparada com uma string fixa sendo que a ordem dos elementos na variável *havePrimitiveType* não é garantida.

Correção do Flaky: Foi utilizado JSONAssert para verificar a igualdade entre Json, já que o JSONAssert desconsidera a ordem dos elementos no Json, tornando o teste determinístico e estável.

Motivo redução de Test Smells: A categoria *Assertion Roulette* ocorre quando em um teste possui mais de um *assertion statement sem uma mensagem descritiva*. A correção do flaky substituiu as chamadas do assertEquals pelo método assertJSONEqual que não leva em consideração a ordem dos elementos do Json. Com essa substituição a ferramenta TsDetect não detectou o smells, desta forma um smells foi reduzido da categoria *Assertion Roulette*, porém o smells ainda persiste só não foi detectado.

Motivo aumento de Test Smells: A categoria *Exception Catching Throwing* ocorre quando um teste contém uma instrução throw ou uma cláusula catch. Na correção do flaky foi adicionado o lançamento da exceção JSONException em 1 teste, com isso a categoria *Exception Catching Throwing* aumentou em 1 a quantidade de smells.

Conclusão: Devido a utilização de outro assert no teste que não pertence a JUnit, a quantidade de smells foi reduzida, porém a correção do flakiness não está relacionada a essa redução.

```
211 @Test
212- public void testKeyValueSchemaInfoToString() {
213     String havePrimitiveType = DefaultImplementation
214         .convertKeyValueSchemaInfoDataToString(KeyValueSchemaInfo
215             .decodeKeyValueSchemaInfo(Schema.KeyValue(Schema.AVRO(Foo.class), Schema.STRING)
216                 .getSchemaInfo()));
217-     assertEquals(havePrimitiveType, KEY_VALUE_SCHEMA_INFO_INCLUDE_PRIMITIVE);
218
219     String notHavePrimitiveType = DefaultImplementation
220         .convertKeyValueSchemaInfoDataToString(KeyValueSchemaInfo
221             .decodeKeyValueSchemaInfo(Schema.KeyValue(Schema.AVRO(Foo.class),
222                 Schema.AVRO(Foo.class)).getSchemaInfo()));
223-     assertEquals(notHavePrimitiveType, KEY_VALUE_SCHEMA_INFO_NOT_INCLUDE_PRIMITIVE);
224 }
```

```
212 @Test
213+ public void testKeyValueSchemaInfoToString() throws JSONException {
214     String havePrimitiveType = DefaultImplementation
215         .convertKeyValueSchemaInfoDataToString(KeyValueSchemaInfo
216             .decodeKeyValueSchemaInfo(Schema.KeyValue(Schema.AVRO(Foo.class), Schema.STRING)
217                 .getSchemaInfo()));
218+     JSONSchemaTest.assertJSONEqual(havePrimitiveType, KEY_VALUE_SCHEMA_INFO_INCLUDE_PRIMITIVE);
219
220     String notHavePrimitiveType = DefaultImplementation
221         .convertKeyValueSchemaInfoDataToString(KeyValueSchemaInfo
222             .decodeKeyValueSchemaInfo(Schema.KeyValue(Schema.AVRO(Foo.class),
223                 Schema.AVRO(Foo.class)).getSchemaInfo()));
224+     JSONSchemaTest.assertJSONEqual(notHavePrimitiveType, KEY_VALUE_SCHEMA_INFO_NOT_INCLUDE_PRIMITIVE);
225 }
```

4. JsonMapperTest

Projeto: vipshop/vjtools

Arquivo: JsonMapperTest.java

PR: <https://github.com/vipshop/vjtools/pull/168>

Categorias que alteram: Assertion Roulette, Exception Catching Throwing

Redução: 1

Aumento: 2

Motivo do Flaky: Os testes toJson(), threeTypeMappers() e jsonp() comparavam a string do Json retornado com uma string fixa. Entretanto, a ordem dos atributos do Json não é garantida e pode ser retornado o objeto em qualquer ordem, o que torna o teste não determinístico.

Correção do Flaky: Foi utilizado JSONAssert para verificar a igualdade entre objetos Json, pois o JSONAssert desconsidera a ordem dos elementos no Json, tornando o teste determinístico e removendo a flakiness.

Motivo redução de Test Smells: A categoria *Assertion Roulette* ocorre quando em um teste possui mais de um *assertion statement sem uma mensagem descritiva*. A correção do flaky substituiu alguns assertion statement pelo método assertJSONEqual que não leva em consideração a ordem dos atributos do Json, assim o teste threeTypeMappers() que possuía 4 asserts passou a ter apenas um, o que reduz smells da categoria *Assertion Roulette*, entretanto os smells continuam acontecendo pois foi apenas substituído um assert (assertThat) por outro (assertJSONEqual) fazendo com que a ferramenta TsDetect não detecte os smells e reduzindo a quantidade.

Motivo aumento de Test Smells: A categoria *Exception Catching Throwing* ocorre quando um teste contém uma instrução throw ou uma cláusula catch. Na correção do flaky foi adicionado o lançamento da exceção JSONException nos testes threeTypeMappers() e jsonp(), com isso a categoria *Exception Catching Throwing* aumentou 2 smells.

Conclusão: Devido a utilização do JSONAssert para comparar os Json, a quantidade de smells foi reduzida pois o JSONAssert não pertence a JUnit, não sendo considerado pela ferramenta como um *assertion statement*. Essa redução de test smells não influencia na correção do flakiness.

```
147- @Test
148- public void threeTypeMappers() {
149-     // 打印全部属性
150-     ObjectMapper normalBinder = ObjectMapper.defaultMapper();
151-     TestBean bean = new TestBean(name:"A");
152-     assertThat(normalBinder.toJson(bean))
153-         .isEqualTo("{\"name\":\"A\",\"defaultValue\":\"hello\",\"nullValue\":null,\"emptyValue\":[]}");
154-
155-     // 不打印nullValue属性
156-     ObjectMapper nonNullMapper = ObjectMapper.nonNullMapper();
157-     assertThat(nonNullMapper.toJson(bean))
158-         .isEqualTo("{\"name\":\"A\",\"defaultValue\":\"hello\",\"emptyValue\":[]}");
159-
160-     // 不打印nullValue与empty的属性
161-     ObjectMapper nonEmptyMapper = ObjectMapper.nonEmptyMapper();
162-     assertThat(nonEmptyMapper.toJson(bean)).isEqualTo("{\"name\":\"A\",\"defaultValue\":\"hello\"}");
163-
164-     TestBean nonEmptyBean = nonEmptyMapper.fromJson("{\"name\":\"A\",\"defaultValue\":\"hello\"}", TestBean.class);
165-     assertThat(nonEmptyBean.getEmptyValue()).isEmpty();
166- }
167-
168- @Test
169- public void jsonp() {
170-     TestBean bean = new TestBean(name:"A");
171-     String jsonp = ObjectMapper.nonEmptyMapper().toJsonP("haha", bean);
172-     assertThat(jsonp).isEqualTo("haha({\"name\":\"A\",\"defaultValue\":\"hello\"})");
173- }
```

```
152- @Test
153- public void threeTypeMappers() throws JSONException {
154-     // 打印全部属性
155-     ObjectMapper normalBinder = ObjectMapper.defaultMapper();
156-     TestBean bean = new TestBean(name:"A");
157-     assertJSONEqual(normalBinder.toJson(bean), s2:"{\"name\":\"A\",\"defaultValue\":\"hello\",\"nullValue\":null,\"emptyValue\":[]}");
158-
159-     // 不打印nullValue属性
160-     ObjectMapper nonNullMapper = ObjectMapper.nonNullMapper();
161-     assertJSONEqual(nonNullMapper.toJson(bean), s2:"{\"name\":\"A\",\"defaultValue\":\"hello\",\"emptyValue\":[]}");
162-
163-     // 不打印nullValue与empty的属性
164-     ObjectMapper nonEmptyMapper = ObjectMapper.nonEmptyMapper();
165-     assertJSONEqual(nonEmptyMapper.toJson(bean), s2:"{\"name\":\"A\",\"defaultValue\":\"hello\"}");
166-
167-     TestBean nonEmptyBean = nonEmptyMapper.fromJson("{\"name\":\"A\",\"defaultValue\":\"hello\"}", TestBean.class);
168-     assertThat(nonEmptyBean.getEmptyValue()).isEmpty();
169- }
170-
171- @Test
172- public void jsonp() throws JSONException{
173-     TestBean bean = new TestBean(name:"A");
174-     String jsonp = ObjectMapper.nonEmptyMapper().toJsonP("haha", bean);
175-     String expected = testJSON.substring(testJSON.indexOf(str:"()x1",testJSON.indexOf(str:"()"));
176-     String test = jsonp.substring(jsonp.indexOf(str:"()x1",jsonp.indexOf(str:"()"));
177-     assertThat(jsonp.replace(test, replacement:"")).isEqualTo(testJSON.replace(expected, replacement:""));
178-     assertJSONEqual(expected, test);
179- }
180- }
```

Sensitive Equality e Exception Catching Throwing

5. DatadogBackendClientTest

Projeto: DataDog/jmeter-datadog-backend-listener

Arquivo: DatadogBackendClientTest.java

PR: <https://github.com/DataDog/jmeter-datadog-backend-listener/pull/33>

Categorias que alteram: Exception Catching Throwing, Sensitive Equality

Redução: 1

Aumento: 1

Motivo do Flaky: A causa da falha do teste é devido à comparação de duas strings JSON. No entanto, a ordem dos campos armazenados em JSONObject não é determinística, o que pode retornar resultados diferentes por execução ao chamar o método toString() em um JSONObject.

Correção do Flaky: A correção realizada foi usar um JSONParser para converter a string em um JSONObject e compará-los, já que a ordem dos elementos não interfere na comparação.

Motivo redução de Test Smells: A categoria *Sensitive Equality* ocorre quando um teste utiliza o método toString() em um assertion statement. A correção do flaky removeu a utilização do método toString() diminuindo um smells da categoria *Sensitive Equality*.

Motivo aumento de Test Smells: A categoria *Exception Catching Throwing* ocorre quando um teste contém uma instrução throw ou uma cláusula catch. Na correção do flaky foi adicionado o lançamento da exceção ParseException em 1 teste, com isso a categoria *Exception Catching Throwing* aumentou em 1 a quantidade de smells.

Conclusão: A correção do test smells Sensitive Equality obriga o desenvolvedor a remover a utilização do método toString(). Nesse teste o flakiness ocorre devido a comparação de duas strings Json, então ao remover a utilização do toString para comparar as strings e passar a utilizar o JsonParser para compara os objetos Json, a correção do test smells é realizada e consequentemente a remoção do flakiness. O smells Exception Catching não tem relação com a correção do flakiness.

```
171     @Test
172-    public void testExtractLogs() {
173        SampleResult result = createDummySampleResult(sampleLabel:"foo");
174        this.client.handleSampleResults(Collections.singletonList(result), context);
175        Assert.assertEquals(1, this.logsBuffer.size());
176        String expectedPayload = "{\"sample_start_time\":1.0,\"response_code\":\"123\",\"headers_size\":0.0,\"sample_label\":\"foo\"}";
177-        Assert.assertEquals(this.logsBuffer.get(0).toString(), expectedPayload);
178    }

173     @Test
174+    public void testExtractLogs() throws ParseException {
175        SampleResult result = createDummySampleResult(sampleLabel:"foo");
176        this.client.handleSampleResults(Collections.singletonList(result), context);
177        Assert.assertEquals(1, this.logsBuffer.size());
178        String expectedPayload = "{\"sample_start_time\":1.0,\"response_code\":\"123\",\"headers_size\":0.0,\"sample_label\":\"foo\",\"late\"}";
179+        JSONParser parser = new JSONParser(JSONParser.MODE_PERMISSIVE);
180+        Assert.assertEquals(this.logsBuffer.get(0), parser.parse(expectedPayload));
181    }
```

Sensitive Equality e Duplicate Assert

6. JSONPath_reverse_test

Projeto: alibaba/fastjson

Arquivo: JSONPath_reverse_test.java

PR: <https://github.com/alibaba/fastjson/pull/3530>

Categorias que alteram: Sensitive Equality, Duplicate Assert

Redução: 1

Aumento: 1

Motivo do Flaky: Os testes existentes são flaky (inconsistentes) porque dependem da ordem dos elementos em um Map.

Correção do Flaky: Para corrigir foi utilizado `SerializerFeature.MapSortField` para forçar a ordem ao converter JSON em string, para que a string convertida seja determinística.

Motivo redução de Test Smells: A categoria *Sensitive Equality* ocorre quando o método `toString` é usado em um teste. Com a utilização do método `toJSONString` junto com o `SerializerFeature.MapSortField` para ordenar o map antes de converter em string o método `toString` foi removido o que ocasionou a redução da categoria *Sensitive Equality*.

Motivo aumento de Test Smells: A categoria *Duplicate Assert* ocorre quando um teste possui mais de um assert com os mesmos parâmetros, com a correção do flaky o teste `_reserve3` sofreu uma alteração no qual é atribuído a variável `text` o valor antes de comparar, ficando com dois asserts iguais mesmo o valor de `text` sendo diferente. Como pode ser observado nas linhas 37 e 39 da correção.

Conclusão: Nesse teste o desenvolvedor parou de utilizar o método `toString` pois a ordem não é garantida, e passou a utilizar um método no qual ordena os elementos do Json antes da comparação, com isso o test smells é corrigido e o flakiness removido. É possível notar que a correção do test smells e o flakiness estão relacionadas. O smells *Duplicate Assert* não possui relação com o flakiness, o desenvolvedor poderia atribuir o resultado da linha 38 da correção a uma nova variável diferente de `"text"` que o smells não iria acontecer, porém o flakiness seria removido.

```
26 public void test_reserve3() throws Exception {
27     JSONObject object = JSON.parseObject("{\"player\":{\"id\":\"1001\",\"name\":\"ljw\",\"age\":\"50\"}}");
28
29     assertEquals("{\"player\":{\"name\":\"ljw\",\"id\":\"1001\"}}", JSONPath.reserveToObject(object, "player.id", "player.name").toString());
30     assertEquals("{\"player\":{\"name\":\"ljw\",\"id\":\"1001\"}}", JSONPath.reserveToObject(object, "player.name", "player.id", "ab.c").toString());
31 }

33 public void test_reserve3() throws Exception {
34     JSONObject object = JSON.parseObject("{\"player\":{\"id\":\"1001\",\"name\":\"ljw\",\"age\":\"50\"}}");
35
36+    String text = JSON.toJSONString(JSONPath.reserveToObject(object, "player.id", "player.name"), SerializerFeature.MapSortField);
37+    assertEquals("{\"player\":{\"id\":\"1001\",\"name\":\"ljw\"}}", text);
38+    text = JSON.toJSONString(JSONPath.reserveToObject(object, "player.name", "player.id", "ab.c"), SerializerFeature.MapSortField);
39+    assertEquals("{\"player\":{\"id\":\"1001\",\"name\":\"ljw\"}}", text);
40 }
```

Sensitive Equality e General Fixture

7. JacksonXmlHandlerTest

Projeto: Apache/Struts

Arquivo: JacksonXmlHandlerTest_a368cb6.java

PR: <https://github.com/apache/struts/pull/500>

Categorias que alteram: General Fixture, Sensitive Equality

Redução: 1

Aumento: 1

Motivo do Flaky: O método `handler.fromObject()` serializa o objeto de maneira não determinística no qual a ordem dos elementos não é garantida, portanto ao fazer a comparação do xml em algum momentos pode falhar.

Correção do Flaky: Foi corrigido comparando o tamanho total do xml e se contém cada elemento de forma separada o que ignora a ordem e remove o flakiness.

Motivo redução de Test Smells: A categoria *Sensitive Equality* ocorre quando um teste utiliza o método `toString()` em um `assertion statement`. A correção do flaky removeu a utilização do método `toString` de dentro do `assertTrue` e atribuiu a variável *actual* ocasionando a não detecção do smells pela ferramenta e diminuindo um smell da categoria *Sensitive Equality*.

Motivo aumento de Test Smells: A categoria *General Fixture* ocorre quando nem todos os campos instanciados no método `setUp` são utilizados por todos os métodos de teste na mesma classe, o que indica que o `setUp` é muito geral, uma desvantagem de ser muito geral é que um trabalho desnecessário está sendo feito quando um método de teste é executado. A correção do flaky atribui algumas variáveis a mais no `setUp` para dividir os elementos do xml, porém essas variáveis são utilizadas apenas por um dos dois testes dessa classe, aumentando os smells da categoria *General Fixture*.

Conclusão: Nesse caso, a correção do test smell pode ocasionar a remoção do flakiness, como o desenvolvedor é obrigado a remover o `toString` do *assertion statement*, a refatoração tende a remover a comparação com strings fixas. Nota-se que o método `toString` ainda é utilizado como visto na linha 68 da correção, porém por não estar dentro do *assertion statement* o TsDetect não identificou o test smells, uma melhoria é necessária no TsDetect para identificar esses casos, mas mesmo com isso a correção desse tipo de smells é capaz de remover o flakiness do código, pois força o desenvolvedor a utilizar outros tipo de comparação no lugar de string fixas.

```
39 public void setUp() throws Exception {
40     super.setUp();
41     xml = "<SimpleBean>" +
42         "<name>Jan</name>" +
43         "<age>12</age>" +
44         "<parents>" +
45         "<parents>Adam</parents>" +
46         "<parents>Ewa</parents>" +
47         "</parents>" +
48         "</SimpleBean>";
49
50     handler = new JacksonXmlHandler();
51     ai = new MockActionInvocation();
52 }
53
54 public void testObjectToXml() throws Exception {
55     // given
56     SimpleBean obj = new SimpleBean();
57     obj.setName("Jan");
58     obj.setAge(12L);
59     obj.setParents(Arrays.asList(...a:"Adam", "Ewa"));
60
61     // when
62     Writer stream = new StringWriter();
63     handler.fromObject(ai, obj, null, stream);
64
65     // then
66     stream.flush();
67     assertEquals(xml, stream.toString());
```

```

39 public void setUp() throws Exception {
40     super.setUp();
41     name = "<name>Jan</name>";
42     age = "<age>12</age>";
43     parents = "<parents>" +
44         "<parents>Adam</parents>" +
45         "<parents>Ewa</parents>" +
46         "</parents>";
47     prefix = "<SimpleBean>";
48     suffix = "</SimpleBean>";
49     xml = prefix + name + age + parents + suffix;
50
51     handler = new JacksonXmlHandler();
52     ai = new MockActionInvocation();
53 }
54
55 public void testObjectToXml() throws Exception {
56     // given
57     SimpleBean obj = new SimpleBean();
58     obj.setName("Jan");
59     obj.setAge(12L);
60     obj.setParents(Arrays.asList(...a:"Adam", "Ewa"));
61
62     // when
63     Writer stream = new StringWriter();
64     handler.fromObject(ai, obj, null, stream);
65
66     // then
67     stream.flush();
68     String actual = stream.toString();
69     assertTrue(actual.length() == xml.length() &&
70         actual.startsWith(prefix) &&
71         actual.contains(name) &&
72         actual.contains(age) &&
73         actual.contains(parents) &&
74         actual.endsWith(suffix));
75 }

```

Conditional Test Logic e Duplicate Assert

8. Issue1584

Projeto: alibaba/fastjson

Arquivo: Issue1584.java

PR: <https://github.com/alibaba/fastjson/pull/3570>

Categorias que alteram: Conditional Test Logic, Duplicate Assert

Redução: 1

Aumento: 1

Motivo do Flaky: O teste é Flaky porque depende da ordenação de pares chave-valor em um Map.

Correção do Flaky: Foi considerado todas as ordenações possíveis do Map para remover as inconsistências (Flaky)

Motivo redução de Test Smells: A categoria *Duplicate Assert* ocorre quando um teste possui mais de um assert com os mesmos parâmetros, no caso desse arquivo de teste como foi refatorado para remover o flakiness, o código `assertEquals("A", entry.getValue());` foi alterado para `assertEquals("A", value);` o que fez o test smells ser removido já que no teste possui outra linha com o mesmo assert.

Motivo aumento de Test Smells: A categoria *Conditional Test Logic* ocorre quando o teste possui condicional, o que pode levar a situações em que o teste falha em detectar defeitos no método de produção, pois as instruções de teste não foram executadas porque uma condição não foi atendida. Após a correção do Flaky foi adicionado uma condicional para verificar se o valor do Map é "A" ou "1" a depender da chave, com isso a quantidade de smells sofreu um aumento nessa categoria.

Conclusão: Nenhuma das alterações no test smells tem relação com a correção do flakiness, pois no caso do *Duplicate Assert*, poderia ter feito apenas atribuição em uma variável para remover os asserts duplicados assim como feito na correção. Já para a categoria *Conditional*

Test Logic a quantidade de smells aumentou.

```
16 public void test_for_issue() throws Exception {
17     ParserConfig config = new ParserConfig();
18
19     String json = "{\"k\":1,\"v\":\"A\"}";
20
21     {
22         Map.Entry entry = JSON.parseObject(json, Map.Entry.class, config);
23         assertEquals("v", entry.getKey());
24         assertEquals("A", entry.getValue());
25     }
26 }
```

```
16 public void test_for_issue() throws Exception {
17     ParserConfig config = new ParserConfig();
18
19     String json = "{\"k\":1,\"v\":\"A\"}";
20
21     {
22         Map.Entry entry = JSON.parseObject(json, Map.Entry.class, config);
23         Object key = entry.getKey();
24         Object value = entry.getValue();
25         assertTrue(key.equals(obj:"v") || key.equals(obj:"k"));
26         if (key.equals(obj:"v")) {
27             assertEquals("A", value);
28         } else {
29             assertEquals(1, value);
30         }
31     }
32 }
```

Diversos

9. PortalRegistryTest

Projeto: networknt/light-4j

Arquivo: PortalRegistryTest_e1e8486.java

PR: <https://github.com/networknt/light-4j/pull/857>

Categorias que alteram: Exception Catching Throwing, General Fixture, Sleepy Test, Unknown Test, IgnoredTest, Magic Number Test

Redução: 5

Aumento: 1

Motivo do Flaky: O resultado dos testes podem ser diferentes a depender da ordem na qual os testes são executados. O teste subAndUnsubService() não remove o registro ao final do teste, o que impacta os testes doRegisterAndAvailable() e discoverService() a depender da ordem de execução. Algo semelhante ocorre com o teste discoverService(), se simplesmente executar os testes não tem como saber que o teste discoverService() possui serviço indisponível.

Correção do Flaky: No teste subAndUnsubService() remover o registro das URLs e no teste discoverService() tornar o serviço indisponível.

Motivo redução de Test Smells: A categoria *Exception Catching Throwing* ocorre quando um teste contém uma instrução throw ou uma cláusula catch. Na PR de correção do flaky foi adicionado a anotação @Ignore no teste subAndUnsubService(), com isso o teste que tinha o lançamento de uma Exception ao ser ignorado diminuiu um smells da categoria *Exception Catching Throwing*.

O mesmo caso acontece com a categoria *General Fixture* que ocorre quando nem todos os campos instanciados no método setUp são utilizados por todos os métodos de teste na mesma classe. Como o teste subAndUnsubService() não utilizava todas as variáveis definidas no setUp, ao ser ignorado foi reduzido um smells da categoria *General Fixture*.

A categoria *Sleepy Test* ocorre quando um teste utiliza o método `Thread.sleep()` o que pode levar a resultados inesperados já que o tempo de processamento pode ser distinto em diferentes dispositivos. Como o teste `subAndUnsubService()` utilizava o método `Thread.sleep()`, ao ser ignorado foi reduzido um smells da categoria *Sleepy Test*.

A categoria *Unknown Test* ocorre quando um teste não contém um único *assertion statement*. Semelhante às categorias descritas anteriormente, como o teste `subAndUnsubService()` foi ignorado, um *Unknown Test* smells foi reduzido, pois o teste ignorado não possuía nenhum *assertion statement*.

A categoria *Magic Number Test* ocorre quando um teste contém números literais no *assert statement* como parâmetros. Não foi identificado a causa específica para essa categoria sofrer alteração, mas ao ignorar o teste `subAndUnsubService()` foi reduzido um smells dessa categoria.

Motivo aumento de Test Smells: A categoria *Ignored Test* ocorre quando um teste contém a anotação `@Ignore`. Esses testes ignorados resultam em overhead, pois adicionam sobrecarga desnecessária em relação ao tempo de compilação e aumentam a complexidade e a compreensão do código. Como o teste `subAndUnsubService()` foi ignorado, a categoria *Ignored Test* aumentou um smells.

Conclusão: Um dos testes foi ignorado, o que não tem relação com a correção do flakiness, porém impactou em todas as alterações do test smells. Desconsiderar esse arquivo nas análises.

```
91+ @Ignore
92 @Test
93 public void subAndUnsubService() throws Exception {
94     //registry.doSubscribe(clientUrl1, null);
95     //registry.doSubscribe(clientUrl2, null);
96
97+     // registry
98     registry.doRegister(serviceUrl1);
99     registry.doRegister(serviceUrl2);
100    registry.doAvailable(null);
101    Thread.sleep(sleepTime);
102+
103+    // unregistry
104+    registry.doUnavailable(null);
105+    Thread.sleep(sleepTime);
106+    registry.doUnregister(serviceUrl1);
107+    registry.doUnregister(serviceUrl2);
108
109    //registry.doUnsubscribe(clientUrl1, null);
110    //registry.doUnsubscribe(clientUrl2, null);
111 }
```