Universidade Federal de Pernambuco

Centro de Informática

Análise qualitativa do estudo sobre a relação entre test smells e flaky tests

Aluno:
Thalisson Moura Tavares (tmt2@cin.ufpe.br)

Orientador:
Leopoldo Motta Teixeira (lmt@cin.ufpe.br)

Coorientador:
Rodrigo dos Santos Lima (rsl@cin.ufpe.br)

Recife, 17 de Setembro de 2023.

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.

```
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\":\"DE871234567812345678
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

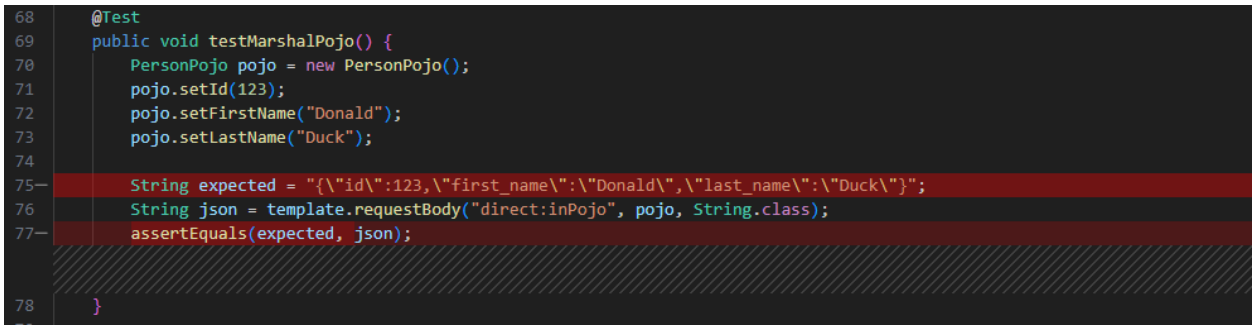
Projeto: Adyen/adyen-java-api-library
Arquivo: SaleToAcquirerDataSerializerTest.java
PR: https://github.com/Adyen/adyen-java-api-library/pull/681
Categorias que alteram: Assertion Roulette
Redução: 1
Aumento: 0
Motivo do Flaky: Comparações diretas entre strings formatadas em JSON podem ocasionalmente 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 detro 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     }
```

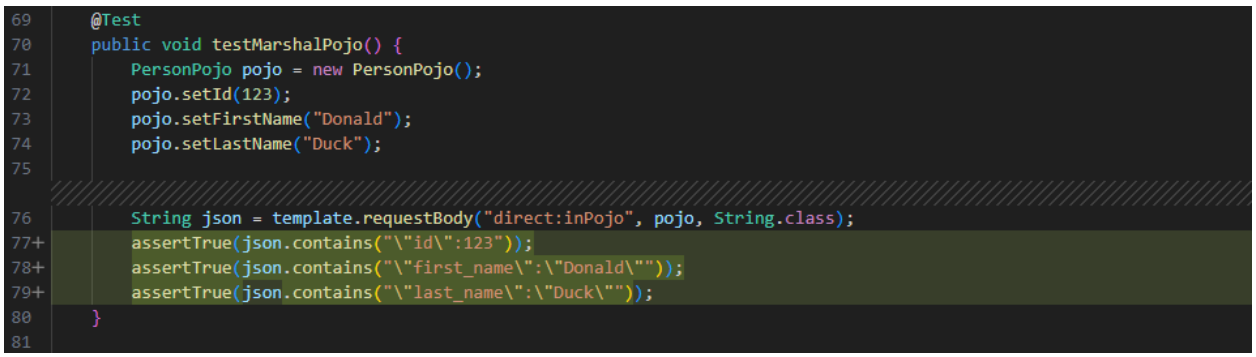
```
102         // test if json string matches
103         String requestJson = PRETTY_PRINT_GSON.toJson(saleToAcquirerData);
104+        assertJsonStringEquals(requestJson, json);
105
106         // test if base64 works
107+        String serialized = saleToAcquirerDataModelAdapter.serialize(saleToAcquirerData, null, null).getAsString();
108+        SaleToAcquirerData saleToAcquirerDataDecoded = new Gson().fromJson(new String(Base64.decodeBase64(serialized)), SaleToAcquirerData.class);
109+        assertEquals(saleToAcquirerData, saleToAcquirerDataDecoded);
110+    }
111+
112+    public static void assertJsonStringEquals(String firstInput, String secondInput) {
113+        JsonParser parser = new JsonParser();
114+        assertEquals(parser.parse(firstInput), parser.parse(secondInput));
115     }
116 }
117
```

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 realizado 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.

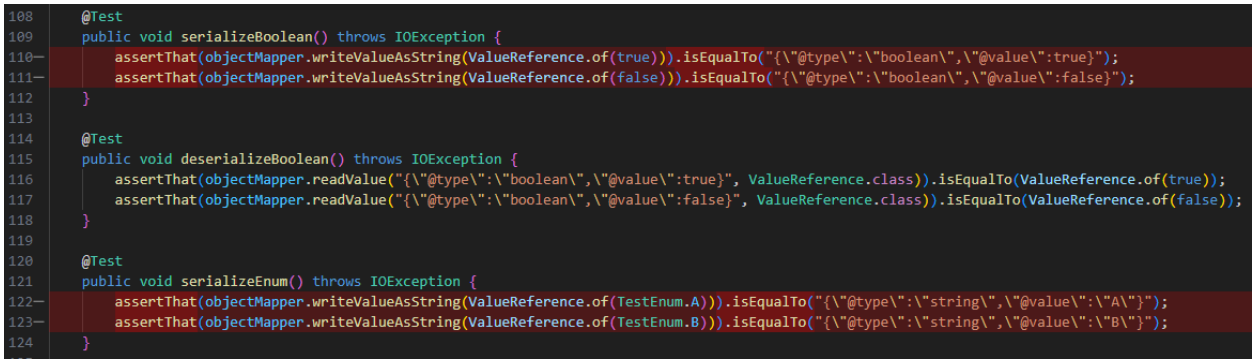
```
229     // [databind#271]
230     public void testUnwrapping()  throws Exception
231     {
232         JsonSchema jsonSchema = MAPPER.generateJsonSchema(UnwrappingRoot.class);
233-        String json = jsonSchema.toString().replaceAll("\"", "'");
234-        String EXP = "{'type':'object',"
235-                +"'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     }
```

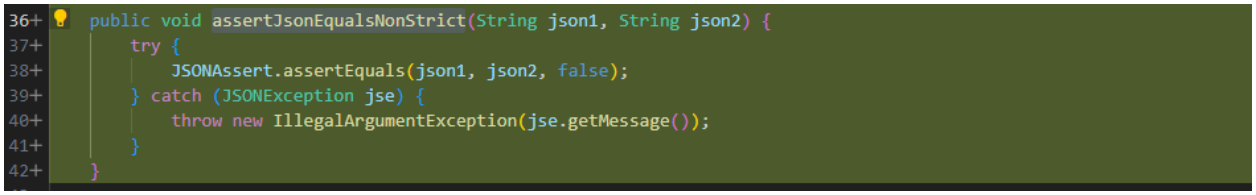
7. ValueReferenceTest

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     }
```

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

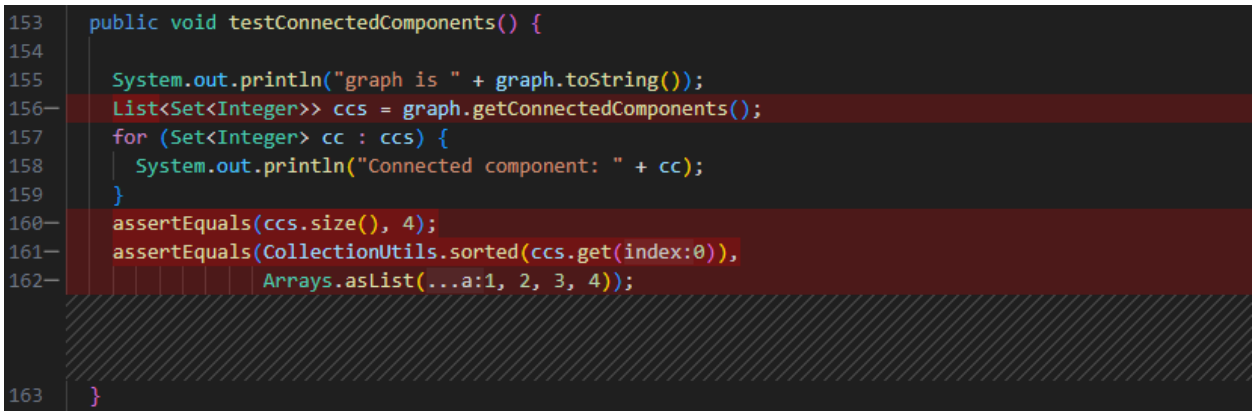
adicionou outros assert no teste getFieldNamesTest() o que ocasionou o aumento de um smells da categoria *Assertion Roulette.*
Conclusão: O aumento de test smells da categoria *Assertion Roulette* não influenciou na remoção do flakiness, o desenvolvedor o desenvolvedor removeu o flakiness ao verificar 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        assertEquals(v, JsonIncludeProperties.Value.from(Bogus.class.getAnnotation(annotationClass:JsonIncludeProperties.class)));
39    }
```

```
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.

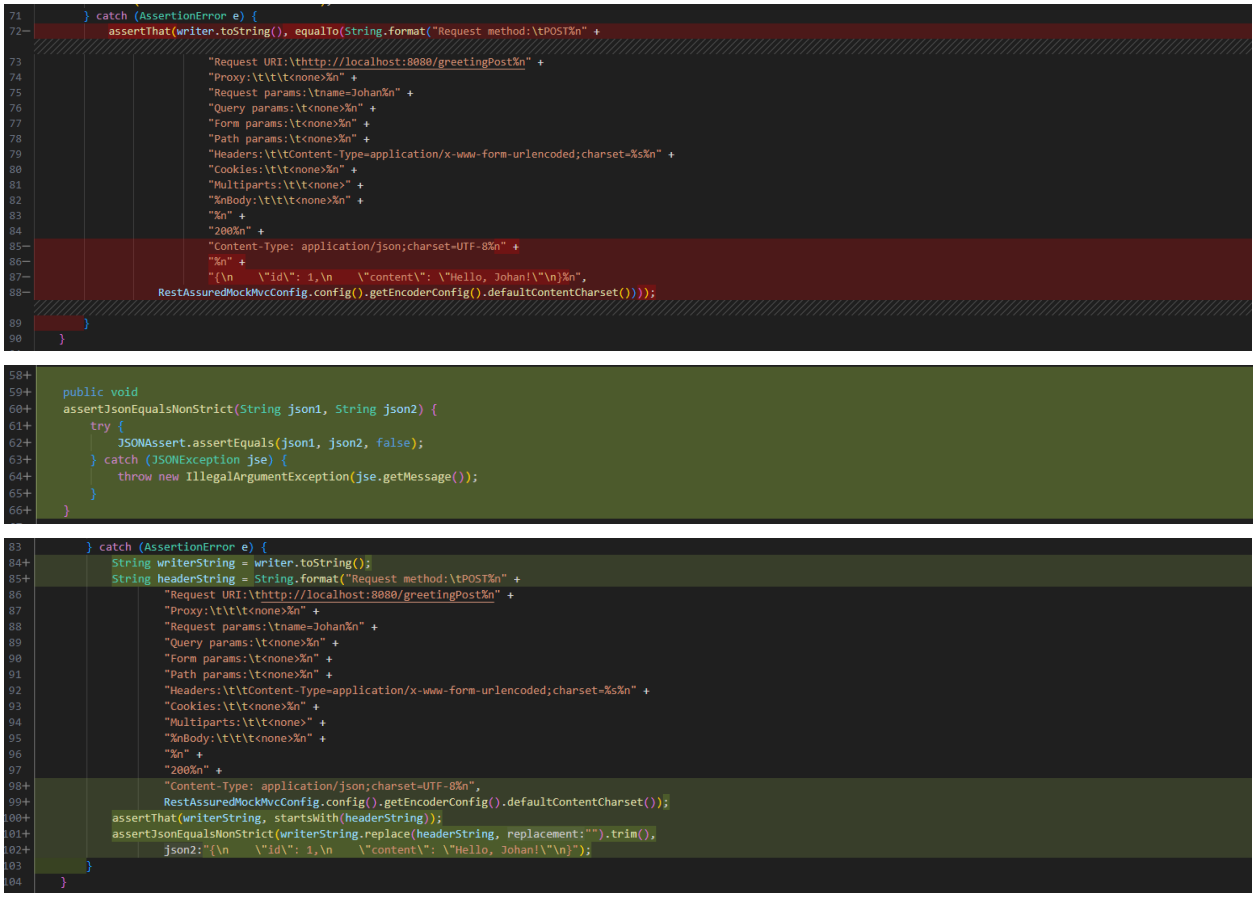
```
@Test
public void testReserialize() {
    BIT b = new BIT();
    byte[] a = new byte[5];
    b.b = Bytes.wrapForRead(a);
    b.c = Bytes.wrapForRead(a);
   // System.out.println(b);

    BIT bit = Marshallable.fromString(BIT.class, "{\n" +
            "b: !!binary AAAAAAA=,\n" +
            "c: !!binary CCCCCCCC,\n" +
            "}");
    assertEquals("!net.openhft.chronicle.wire.BinaryInTextTest$BIT {\n" +
            "  b: !!binary AAAAAAA=,\n" +
            "  c: !!binary CCCCCCCC\n" +
            "}\n", bit.toString());
```

```
@Test
public void testReserialize() {
    BIT b = new BIT();
    byte[] a = new byte[5];
    b.b = Bytes.wrapForRead(a);
    b.c = Bytes.wrapForRead(a);
   // System.out.println(b);

    BIT bit = Marshallable.fromString(BIT.class, "{\n" +
            "b: !!binary AAAAAAA=,\n" +
            "c: !!binary CCCCCCCC,\n" +
            "}");
    String bitToString = bit.toString();
    assertTrue(bitToString.equals("!net.openhft.chronicle.wire.BinaryInTextTest$BIT {\n" +
            "  b: !!binary AAAAAAA=,\n" +
            "  c: !!binary CCCCCCCC\n" +
            "}\n") ||
            bitToString.equals("!net.openhft.chronicle.wire.BinaryInTextTest$BIT {\n" +
                    "  c: !!binary CCCCCCCC,\n" +
                    "  b: !!binary AAAAAAA=\n" +
                    "}\n"));
}
```

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.



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.

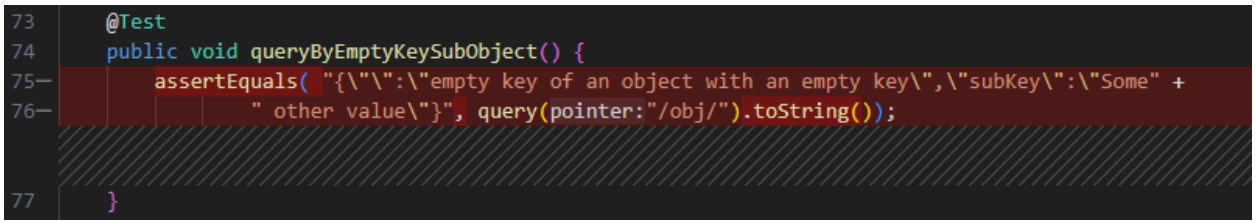
```
184   @Test
185   public void testAddRemove() {
186     CollectionValuedMap<Integer, Integer> fooMap = new CollectionValuedMap<>();
187     for (int i = 0; i < 4; i++) {
188       for (int j = 0; j < 4; j++) {
189         fooMap.add(new Integer(i), new Integer(j));
190       }
191     }
192     fooMap.remove(new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

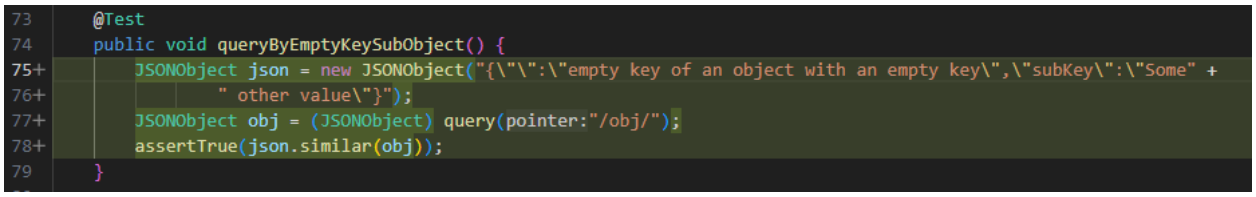
```
187   @Test
188   public void testAddRemove() {
189     CollectionValuedMap<Integer, Integer> fooMap = new CollectionValuedMap<>();
190+    CollectionValuedMap<Integer, Integer> expectedMap = new CollectionValuedMap<>();
191     for (int i = 0; i < 4; i++) {
192       for (int j = 0; j < 4; j++) {
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.

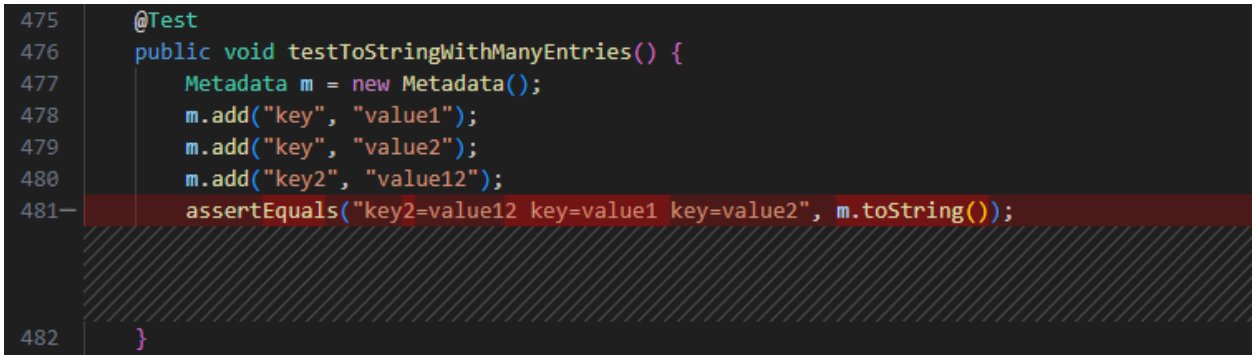




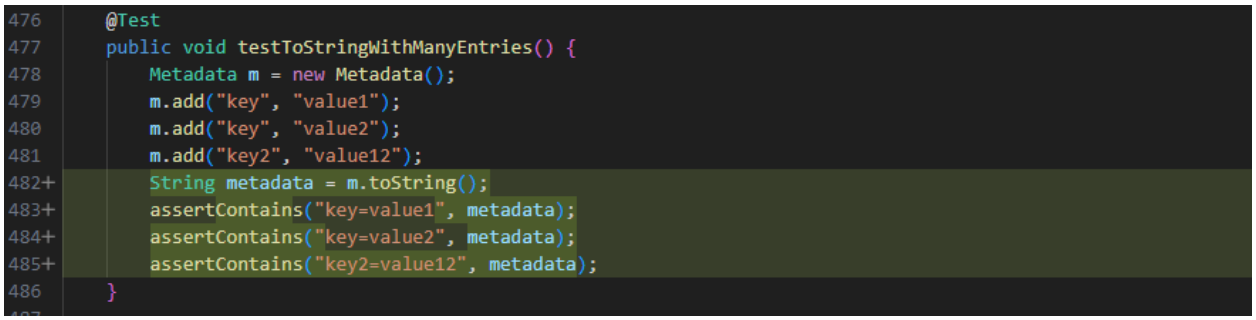
9. TestMetadata

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 outros 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        Assert.assertEquals(HugeType.INDEX_LABEL.string(), indexLabel.type());
49    }
```

```
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

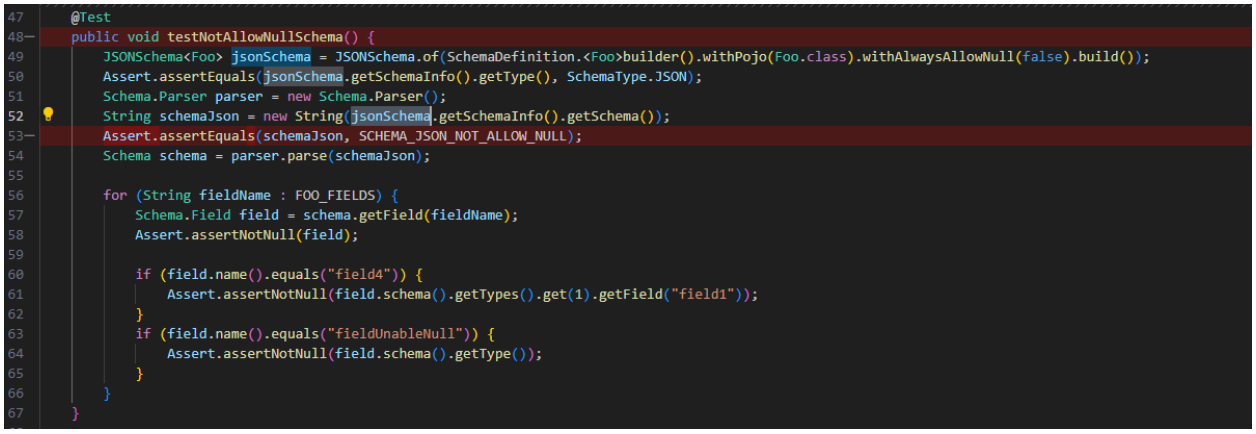
```
145     @Test
146-    public void testNotAllowNullSchema() {
147         AvroSchema<Foo> avroSchema = AvroSchema.of(SchemaDefinition.<Foo>builder().withPojo(Foo.class).withAlwaysAllowNull(false).build());
148         assertEquals(avroSchema.getSchemaInfo().getType(), SchemaType.AVRO);
149         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+
147+    public void assertJSONEquals(String s1, String s2) throws JSONException {
148+        JSONAssert.assertEquals(s1, s2, false);
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+        assertJSONEquals(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     }
```

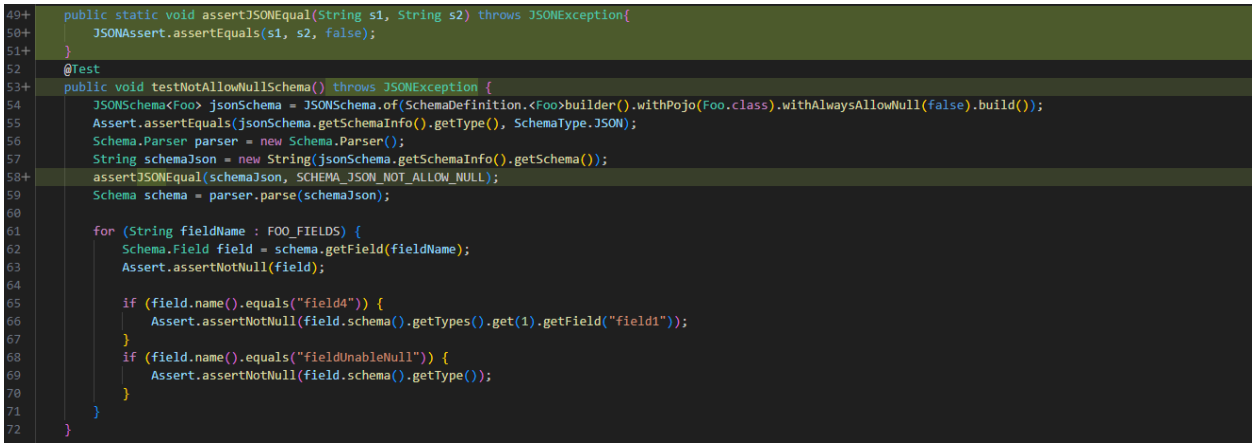
3. JSONSchemaTest

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    }
```

```
67    @Test
68    public void assertGetCurrentShardingTotalCountIfNull() {
69        assertThat(JobRegistry.getInstance().getCurrentShardingTotalCount("exist_job_instance"), is(0));
70    }
71
72    @Test
73+   public void assertGetCurrentShardingTotalCountIfNotNull() throws NoSuchFieldException {
74        JobRegistry.getInstance().setCurrentShardingTotalCount("exist_job_instance", 10);
75        assertThat(JobRegistry.getInstance().getCurrentShardingTotalCount("exist_job_instance"), is(10));
76+       ReflectionUtils.setFieldValue(JobRegistry.getInstance(), "instance", null);
77    }
```

5. Metadata2HeaderFeignInterceptorTest

Projeto: Tencent/spring-cloud-tencent
Arquivo: Metadata2HeaderFeignInterceptorTest.java
PR: https://github.com/Tencent/spring-cloud-tencent/pull/51
Categorias que alteram: Exception Catching Throwing
Redução: 0
Aumento: 1
Motivo do Flaky: O teste pode falhar com base na ordem dos campos em uma string JSON, visto que a comparação é feita com uma string fixa e os atributos do Json podem variar a ordem.
Correção do Flaky: Para ignorar a ordem dos campos, foi utilizado Jackson para analisar as strings JSON. Então, esses objetos JsonNode podem ser comparados diretamente, removendo a flakiness do teste.
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 JsonProcessingException em 1 teste, com isso a categoria *Exception Catching Throwing* aumentou um smells.
Conclusão: Verificar conclusão geral da categoria. Flaky Analysis - Conclusão Thalisson

```
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        Assertions.assertThat(metadataLocalProperties.getContent().get("a")).isEqualTo("1");
69        Assertions.assertThat(metadataLocalProperties.getContent().get("b")).isEqualTo("2");
70        Assertions.assertThat(MetadataContextHolder.get().getTransitiveCustomMetadata("a")).isEqualTo("11");
71        Assertions.assertThat(MetadataContextHolder.get().getTransitiveCustomMetadata("b")).isEqualTo("22");
72        Assertions.assertThat(MetadataContextHolder.get().getTransitiveCustomMetadata("c")).isEqualTo("33");
73    }
```

```
65    @Test
66    public void test1() throws JsonProcessingException {
67        String metadata = testFeign.test();
68        ObjectMapper mapper = new ObjectMapper();
69        Assertions.assertThat(mapper.readTree(metadata)).isEqualTo(mapper.readTree("{\"a\":\"11\",\"b\":\"22\",\"c\":\"33\"}{\"LOCAL_SERVICE\":\"test"
70                + "\",\"LOCAL_PATH\":\"/test\",\"LOCAL_NAMESPACE\":\"default\"}"));
71        Assertions.assertThat(metadataLocalProperties.getContent().get("a")).isEqualTo("1");
72        Assertions.assertThat(metadataLocalProperties.getContent().get("b")).isEqualTo("2");
73        Assertions.assertThat(MetadataContextHolder.get().getTransitiveCustomMetadata("a")).isEqualTo("11");
74        Assertions.assertThat(MetadataContextHolder.get().getTransitiveCustomMetadata("b")).isEqualTo("22");
75        Assertions.assertThat(MetadataContextHolder.get().getTransitiveCustomMetadata("c")).isEqualTo("33");
76    }
```

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 assertEqual 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.

```java
@Test public void findingAnnotationsRecursively() {
    Method method =
        findMethod(this.getClass(), "withMarker", String.class);

    List<Annotation> annotations =
        allAnnotations(method.getParameters()[0]);

    assertEquals(4, annotations.size());
    assertEquals(X.class, annotations.get(index:0).annotationType());
    assertEquals(Y.class, annotations.get(index:1).annotationType());
    assertEquals(Z.class, annotations.get(index:2).annotationType());


    assertEquals(W.class, annotations.get(index:3).annotationType());


}
```

```java
@Test public void findingAnnotationsRecursively() {
    Method method =
        findMethod(this.getClass(), "withMarker", String.class);

    List<Annotation> annotations =
        allAnnotations(method.getParameters()[0]);

    assertEquals(4, annotations.size());
    assertEquals(X.class, annotations.get(index:0).annotationType());
    assertEquals(Y.class, annotations.get(index:1).annotationType());
    assertTrue(Z.class.equals(annotations.get(index:2).annotationType()) ||
        W.class.equals(annotations.get(index:2).annotationType()));
    if (Z.class.equals(annotations.get(index:2).annotationType())) {
        assertEquals(W.class, annotations.get(index:3).annotationType());
    } else {
        assertEquals(Z.class, annotations.get(index:3).annotationType());
    }
}
```

Conclusão da categoria Conditional Test Logic

Quantidade de testes insuficientes para uma conclusão.

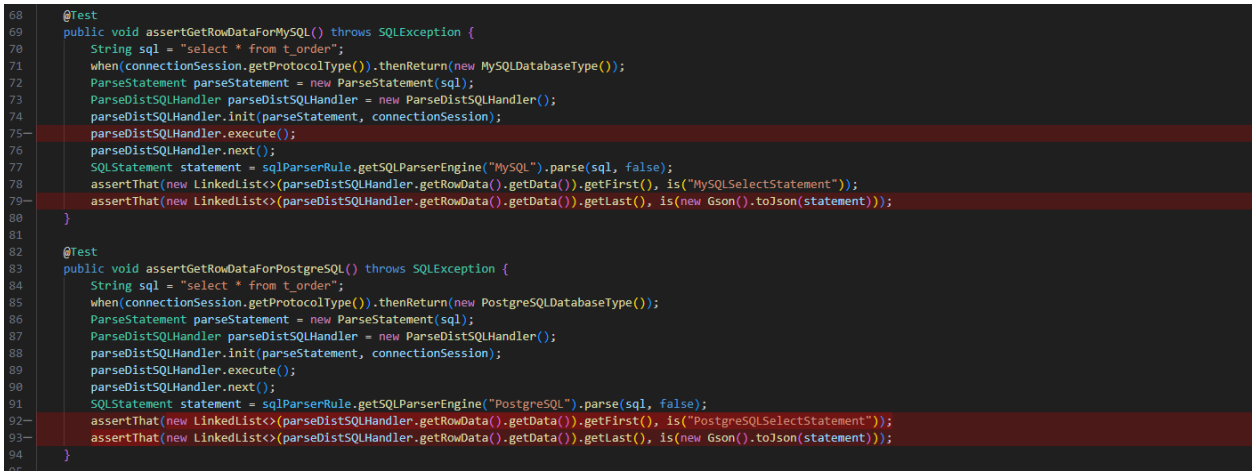
Multiple Categories

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.



```
    @Test
    public void assertGetRowDataForMySQL() throws SQLException {
        String sql = "select * from t_order";
        when(connectionSession.getProtocolType()).thenReturn(new MySQLDatabaseType());
        ParseStatement parseStatement = new ParseStatement(sql);
        ParseDistSQLHandler parseDistSQLHandler = new ParseDistSQLHandler();
        parseDistSQLHandler.init(parseStatement, connectionSession);
        parseDistSQLHandler.execute();
        parseDistSQLHandler.next();
        SQLStatement statement = sqlParserRule.getSQLParserEngine("MySQL").parse(sql, false);
        assertThat(new LinkedList<>(parseDistSQLHandler.getRowData().getData()).getFirst(), is("MySQLSelectStatement"));
        assertThat(new LinkedList<>(parseDistSQLHandler.getRowData().getData()).getLast(), is(new Gson().toJson(statement)));
    }

    @Test
    public void assertGetRowDataForPostgreSQL() throws SQLException {
        String sql = "select * from t_order";
        when(connectionSession.getProtocolType()).thenReturn(new PostgreSQLDatabaseType());
        ParseStatement parseStatement = new ParseStatement(sql);
        ParseDistSQLHandler parseDistSQLHandler = new ParseDistSQLHandler();
        parseDistSQLHandler.init(parseStatement, connectionSession);
        parseDistSQLHandler.execute();
        parseDistSQLHandler.next();
        SQLStatement statement = sqlParserRule.getSQLParserEngine("PostgreSQL").parse(sql, false);
        assertThat(new LinkedList<>(parseDistSQLHandler.getRowData().getData()).getFirst(), is("PostgreSQLSelectStatement"));
        assertThat(new LinkedList<>(parseDistSQLHandler.getRowData().getData()).getLast(), is(new Gson().toJson(statement)));
    }
```

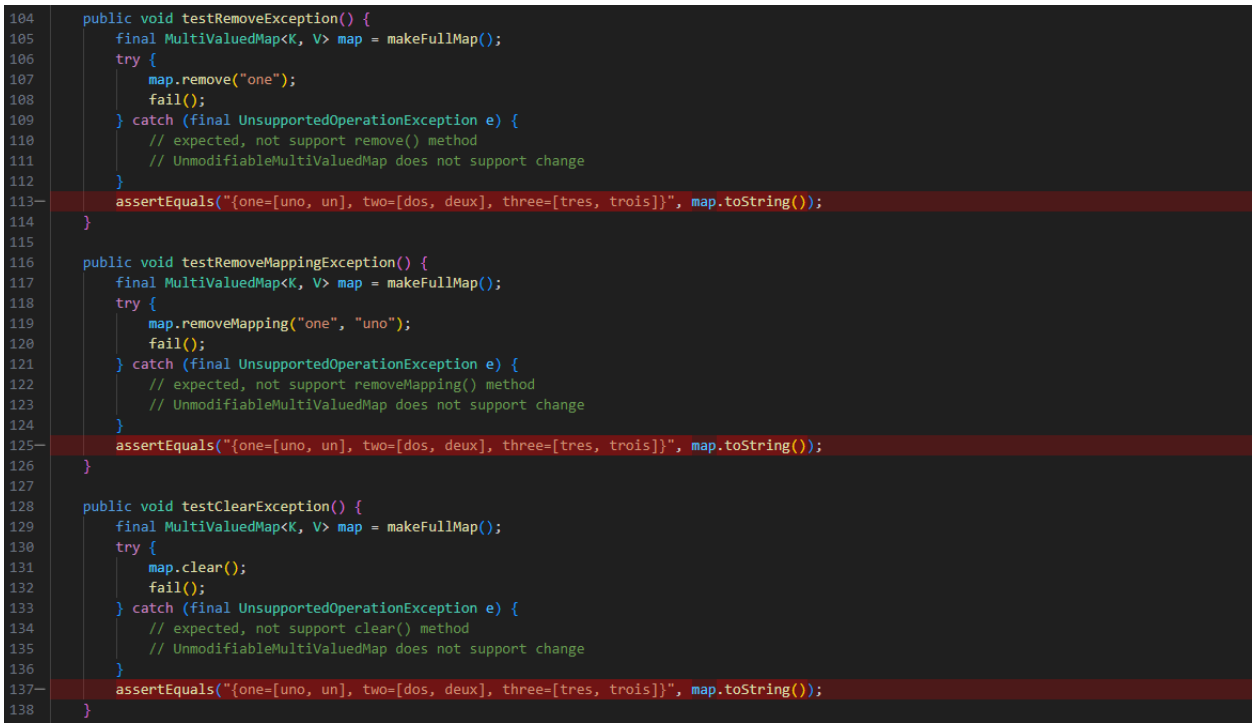
```
    @Test
    public void assertGetRowDataForMySQL() throws SQLException {
        String sql = "select * from t_order";
        when(connectionSession.getProtocolType()).thenReturn(new MySQLDatabaseType());
        ParseStatement parseStatement = new ParseStatement(sql);
        ParseDistSQLHandler parseDistSQLHandler = new ParseDistSQLHandler();
        parseDistSQLHandler.init(parseStatement, connectionSession);
        parseDistSQLHSandler.execute();
        parseDistSQLHandler.next();
        SQLStatement statement = sqlParserRule.getSQLParserEngine("MySQL").parse(sql, false);
        assertThat(new LinkedList<>(parseDistSQLHandler.getRowData().getData()).getFirst(), is("MySQLSelectStatement"));
        assertThat(JsonParser.parseString(new LinkedList<>(parseDistSQLHandler.getRowData().getData()).getLast().toString()), is(JsonParser.parseString(new Gson().toJson(statement))));
    }

    @Test
    public void assertGetRowDataForPostgreSQL() throws SQLException {
        String sql = "select * from t_order";
        when(connectionSession.getProtocolType()).thenReturn(new PostgreSQLDatabaseType());
        ParseStatement parseStatement = new ParseStatement(sql);
        ParseDistSQLHandler parseDistSQLHandler = new ParseDistSQLHandler();
        parseDistSQLHandler.init(parseStatement, connectionSession);
        parseDistSQLHandler.execute();
        parseDistSQLHandler.next();
        SQLStatement statement = sqlParserRule.getSQLParserEngine("PostgreSQL").parse(sql, false);
        assertThat(JsonParser.parseString(new LinkedList<>(parseDistSQLHandler.getRowData().getData()).getLast().toString()), is(JsonParser.parseString(new Gson().toJson(statement))));

    }
```

2. UnmodifiableMultiValuedMapTest

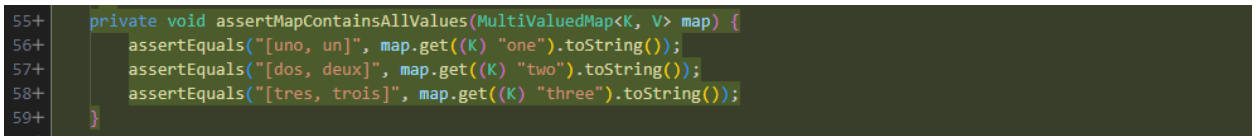
Projeto: apache/commons-collections
Arquivo: UnmodifiableMultiValuedMapTest.java
PR: https://github.com/apache/commons-collections/pull/190
Categorias que alteram: Assertion Roulette, Sensitive Equality
Redução: 6
Aumento: 0
Motivo do Flaky: Várias funções no arquivo utiliza assertEquals entre uma string 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,* entretando 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 substituido 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         JsonMapper normalBinder = JsonMapper.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         JsonMapper nonNullMapper = JsonMapper.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         JsonMapper nonEmptyMapper = JsonMapper.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 = JsonMapper.nonEmptyMapper().toJsonP("haha", bean);
172-        assertThat(jsonp).isEqualTo("haha({\"name\":\"A\",\"defaultValue\":\"hello\"})");
```

```
152     @Test
153+    public void threeTypeMappers() throws JSONException {
154         // 打印全部属性
155         JsonMapper normalBinder = JsonMapper.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         JsonMapper nonNullMapper = JsonMapper.nonNullMapper();
161+        assertJSONEqual(nonNullMapper.toJson(bean),s2:"{\"name\":\"A\",\"defaultValue\":\"hello\",\"emptyValue\":[]}");
162
163         // 不打印nullValue与empty的属性
164         JsonMapper nonEmptyMapper = JsonMapper.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 = JsonMapper.nonEmptyMapper().toJsonP("haha", bean);
175+        String testJSON="haha({\"name\":\"A\",\"defaultValue\":\"hello\"})";
176+        String expected = testJSON.substring(testJSON.indexOf(str:"(")+1,testJSON.indexOf(str:")"));
177+        String test = jsonp.substring(jsonp.indexOf(str:"(")+1, jsonp.indexOf(str:")"));
178+        assertThat(jsonp.replace(test,replacement:"")).isEqualTo(testJSON.replace(expected,replacement:""));
179+        assertJSONEqual(expected, test);
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_lab
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 mesmo 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.

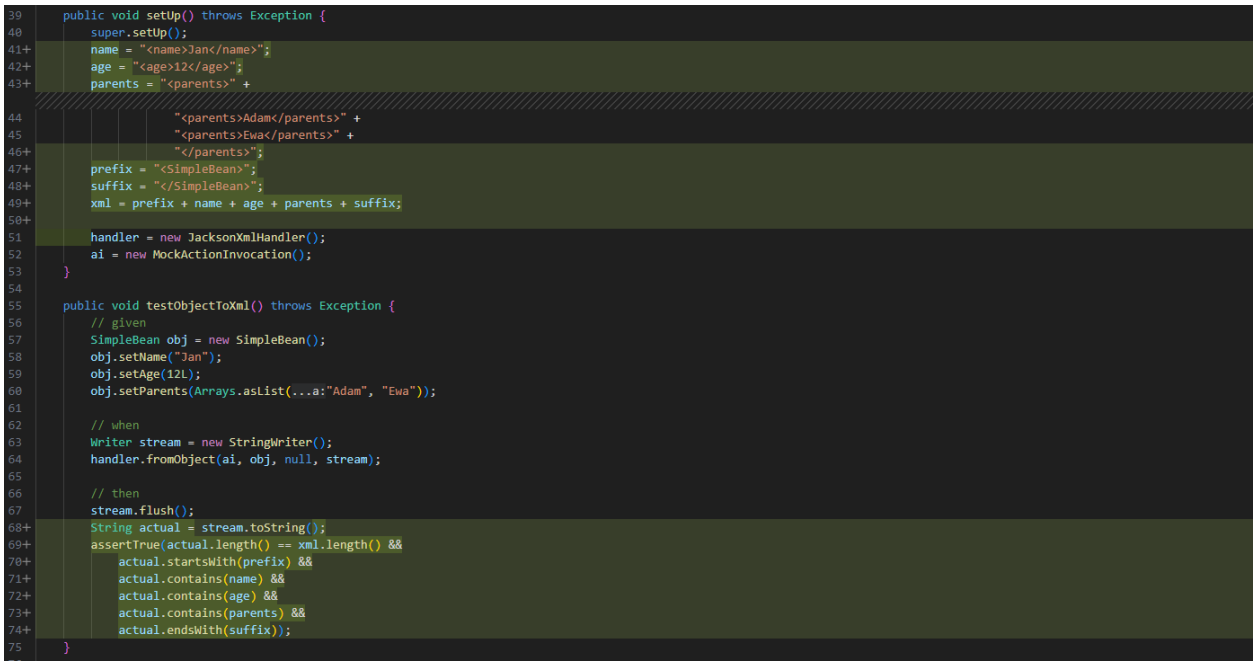
Sensitive Equality e General Fixture

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

```
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          }
```

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.*

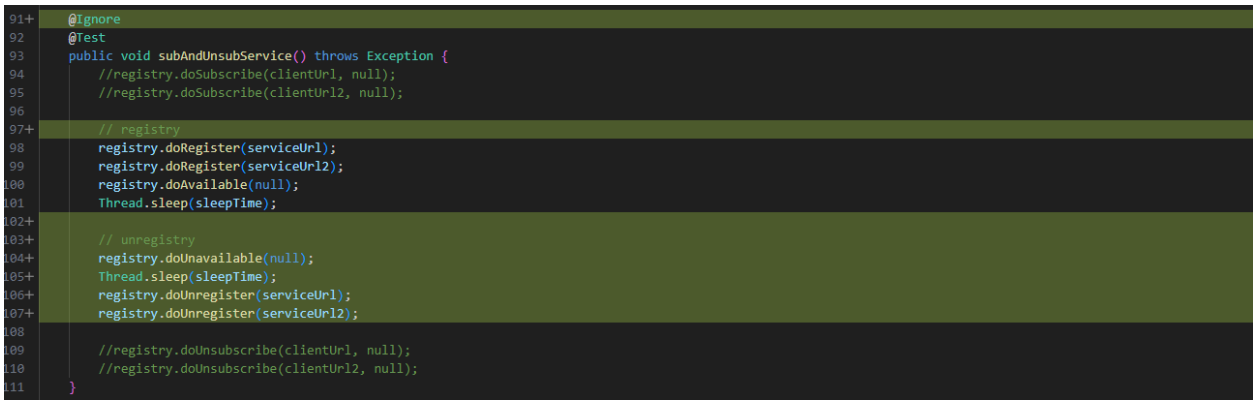
A categoria *Sleepy Test* ocorre quando um teste utiliza o método Thread.sleep() o que pode levar a resultados inesperados já que o tempo de processamento pode ser distinto em diferentes dispositivos. Como o teste subAndUnsubService() utilizava o método Thread.sleep(), ao ser ignorado foi reduzido um 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(clientUrl, null);
95         //registry.doSubscribe(clientUrl2, null);
96
97+        // registry
98         registry.doRegister(serviceUrl);
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(serviceUrl);
107+        registry.doUnregister(serviceUrl2);
108
109        //registry.doUnsubscribe(clientUrl, null);
110        //registry.doUnsubscribe(clientUrl2, null);
111    }
```