Template – Test Smells Refactoring Experiment

Name: Wilhelm de Sousa Steins - 495961
Project: https://github.com/gabrielmaia2/commons-lang

Template – Test Smells Refactoring Experiment:

Assertion Roulette:

```java
@Test
public void testEquivalence() {
    // assertTrue(AnnotationUtils.equals(field1.getAnnotation(TestAnnotation.class), field2.getAnnotation(TestAnnotation.class)));
    // assertTrue(AnnotationUtils.equals(field2.getAnnotation(TestAnnotation.class), field1.getAnnotation(TestAnnotation.class)));
    assertTrue("Identify the equality of field1 with field2", AnnotationUtils.equals(field1.getAnnotation(TestAnnotation.class), field2.getAnnotation(TestAnnotation.class)));
    assertTrue("Identify the equality of field2 with field1", AnnotationUtils.equals(field2.getAnnotation(TestAnnotation.class), field1.getAnnotation(TestAnnotation.class)));
}

@Test
public void testSameInstance() {
    // assertTrue(AnnotationUtils.equals(field1.getAnnotation(TestAnnotation.class), field1.getAnnotation(TestAnnotation.class)));
    assertTrue("Identify the equality of field1 with field2", AnnotationUtils.equals(field1.getAnnotation(TestAnnotation.class), field1.getAnnotation(TestAnnotation.class)));
}

@Test
public void testNonEquivalentAnnotationsOfSameType() {
    // assertFalse(AnnotationUtils.equals(field1.getAnnotation(TestAnnotation.class), field3.getAnnotation(TestAnnotation.class)));
    // assertFalse(AnnotationUtils.equals(field3.getAnnotation(TestAnnotation.class), field1.getAnnotation(TestAnnotation.class)));
    assertFalse("Identify the equality of field1 with field3", AnnotationUtils.equals(field1.getAnnotation(TestAnnotation.class), field3.getAnnotation(TestAnnotation.class)));
    assertFalse("Identify the equality of field3 with field1", AnnotationUtils.equals(field3.getAnnotation(TestAnnotation.class), field1.getAnnotation(TestAnnotation.class)));
}
```

A parte comentada era como estava antes de consertar o Test Smell.

Duplicate Asserton:
Como era antes:

```java
@Test
public void testContains_Char() {
    CharRange range = CharRange.is('c');
    assertFalse(range.contains('b'));
    assertTrue(range.contains('c'));
    assertFalse(range.contains('d'));
    assertFalse(range.contains('e'));

    range = CharRange.isIn('c', 'd');
    assertFalse(range.contains('b'));
    assertTrue(range.contains('c'));
    assertTrue(range.contains('d'));
    assertFalse(range.contains('e'));

    range = CharRange.isIn('d', 'c');
    assertFalse(range.contains('b'));
    assertTrue(range.contains('c'));
    assertTrue(range.contains('d'));
    assertFalse(range.contains('e'));

    range = CharRange.isNotIn('c', 'd');
    assertTrue(range.contains('b'));
    assertFalse(range.contains('c'));
    assertFalse(range.contains('d'));
    assertTrue(range.contains('e'));
    assertTrue(range.contains((char) 0));
    assertTrue(range.contains(Character.MAX_VALUE));
}
```

Como ficou após a refatoração:

```java
@Test
public void testContains_Charx1() {
    CharRange range = CharRange.is('c');
    assertFalse(range.contains('b'));
    assertTrue(range.contains('c'));
    assertFalse(range.contains('d'));
    assertFalse(range.contains('e'));
}

@Test
public void testContains_Charx2() {
    CharRange range = CharRange.is('c', 'd');
    assertFalse(range.contains('b'));
    assertTrue(range.contains('c'));
    assertTrue(range.contains('d'));
    assertFalse(range.contains('e'));
}

@Test
public void testContains_Charx3() {
    CharRange range = CharRange.isIn('d', 'c');
    assertFalse(range.contains('b'));
    assertTrue(range.contains('c'));
    assertTrue(range.contains('d'));
    assertFalse(range.contains('e'));
}

@Test
public void testContains_Charx4() {
    CharRange range = CharRange.isNotIn('c', 'd');
    assertTrue(range.contains('b'));
    assertFalse(range.contains('c'));
    assertFalse(range.contains('d'));
    assertTrue(range.contains('e'));
    assertTrue(range.contains((char) 0));
    assertTrue(range.contains(Character.MAX_VALUE));
}
```

My main difficulties in removing these anomalies were:

Os Eager Tests são bem complicados de serem refatorados e concertados, pensando que alguns funcionam bem da forma que estão criados e a refatoração levou muito trabalho.

I'm using the following refactoring methods to remove test smells:

Eager Test - refatorar em uma ou mais funções;

Assertion Roulette - correção na comparação dos valores, para comparar com valor

padrão ao invés de ser necessário mudar o tipo do valor; Duplicate Assertion - refatorar em uma ou mais funções;
From (0 to 10), how harmful is this test smell to the system? It's because ? Eager Test - 5, depende muito realmente do tamanho e da complexidade do test, pois se for pequeno é bem irrelevante, demoraria tempo tentando, que poderia ser utilizado em outra atividade mais necessária; Assertion Roulette - 5, mesmo que tenha valores mudados, ainda é funcional que a formal que foi criado; Duplicate Assertion - 7, necessário de ser feito para que o código fique mais legível e evitar duplicação de código;