Trabalho final

Refatoração de Tests Smells

Qualidade de Software 2022.2

Equipe:
Ewaldo Junior
Isânio Vitor
Odimar Falcão

Sumário

- Entrega 1
  - Assertion Roulette
  - Eager Test
  - Duplicate Assertion
- Entrega 2
  - Ignored Test
  - Lazy Test
  - Magic Number Test
- Entrega 3
  - Constructor Inicialization
  - Print Statment
  - Unknown Test

Entrega 1

Assertion Roulette

Quantidade de test smells refatorados:

5 test smells refatorados;

Técnica de refatoração para remoção deste test smell:

Incluindo uma explicação para cada assert detectado;

Quantidade de test smells removidos:

5 test smells removidos;

Entrega 1

Assertion Roulette

Refatorações

```
assertTrue(Files.exists(tempDirPath.resolve("f1")));
```

```
assertTrue(Files.exists(tempDirPath.resolve("f1")), "verifica se existe o f1");
```

Entrega 1

Assertion Roulette

Refatorações

```
assertTrue(Files.exists(tempDirPath.resolve("dir")));
```

```
assertTrue(Files.exists(tempDirPath.resolve("dir")), "verifica se existe o dir");
```

Entrega 1

Assertion Roulette

Refatorações

```
assertTrue(Files.exists(targetDir.resolve("dirs-a-file-size-1")));
```

```
assertTrue(Files.exists(targetDir.resolve("dirs-a-file-size-1")), "verifica se existe dirs-a-file-size-1");
```

Entrega 1

Assertion Roulette

Refatorações

```
assertTrue(Files.exists(targetDir.resolve("dirs-a-file-size-1")));
```

```
assertTrue(Files.exists(targetDir.resolve("dirs-a-file-size-1")), "verifica se existe dirs-a-file-size-1");
```

Entrega 1

Assertion Roulette

Refatorações

```
assertTrue(Files.exists(targetDir.resolve("dir")));
```

```
assertTrue(Files.exists(targetDir.resolve("dir")), "verifica se existe dir ");
```

Entrega 1

Eager Test

Quantidade de test smells refatorados:

5 test smells refatorados;

Técnica de refatoração para remoção deste test smell:

Separar os asserts que testam um objeto com diferentes métodos em classes de teste diferentes;

Quantidade de test smells removidos:

5 test smells removidos;

Entrega 1

Eager Test

Refatorações

```java
@Test
public void testFileCleanerDirectory() throws Exception {
    TestUtils.createFile(testFile, 100);
    assertTrue(testFile.exists());
    assertTrue(tempDirFile.exists());

    Object obj = new Object();
    assertEquals(0, theInstance.getTrackCount());
    theInstance.track(tempDirFile, obj);
    assertEquals(1, theInstance.getTrackCount());

    obj = null;

    waitUntilTrackCount();

    assertEquals(0, theInstance.getTrackCount());
    assertTrue(testFile.exists());  // not deleted, as dir
    assertTrue(testFile.getParentFile().exists());  // not
}
```

```java
@Test
public void testFileCleanerDirectory() throws Exception {
    TestUtils.createFile(testFile, 100);
    assertTrue(testFile.exists());
    assertTrue(tempDirFile.exists());

    Object obj = new Object();
    assertEquals(0, theInstance.getTrackCount());

    obj = null;

    waitUntilTrackCount();

    assertTrue(testFile.exists());  // not deleted, as dir
    assertTrue(testFile.getParentFile().exists());  // not
}
```

```java
@Test // testFileCleanerDirectory: Eager Test refactored
public void testFileCleanerDirectory2() throws Exception {
    TestUtils.createFile(testFile, 100);
    assertTrue(testFile.exists());
    assertTrue(tempDirFile.exists());

    Object obj = new Object();
    assertEquals(1, theInstance.getTrackCount());

    obj = null;

    waitUntilTrackCount();

    assertTrue(testFile.exists());  // not deleted, as dir
    assertTrue(testFile.getParentFile().exists());  // not
}
```

Entrega 1

Eager Test

Refatorações

```java
@Test
public void testFileCleanerDirectory() throws Exception {
    TestUtils.createFile(testFile, 100);
    assertTrue(testFile.exists());
    assertTrue(tempDirFile.exists());

    Object obj = new Object();
    assertEquals(0, theInstance.getTrackCount());
    theInstance.track(tempDirFile, obj);
    assertEquals(1, theInstance.getTrackCount());

    obj = null;

    waitUntilTrackCount();

    assertEquals(0, theInstance.getTrackCount());
    assertTrue(testFile.exists());  // not deleted, as dir
    assertTrue(testFile.getParentFile().exists());  // not
}
```

```java
@Test // testFileCleanerDirectory: Eager Test refactored
public void testFileCleanerDirectory3() throws Exception {
    TestUtils.createFile(testFile, 100);
    assertTrue(testFile.exists());
    assertTrue(tempDirFile.exists());

    Object obj = new Object();

    obj = null;

    waitUntilTrackCount();

    assertEquals(0, theInstance.getTrackCount());
    assertTrue(testFile.exists());  // not deleted, as dir
    assertTrue(testFile.getParentFile().exists());  // not
}
```

```java
@Test // testFileCleanerDirectory: Eager Test refactored
public void testFileCleanerDirectory4() throws Exception {
    TestUtils.createFile(testFile, 100);
    assertTrue(testFile.exists());
    assertTrue(tempDirFile.exists());

    Object obj = new Object();
    theInstance.track(tempDirFile, obj);

    obj = null;

    waitUntilTrackCount();

    assertTrue(testFile.exists());  // not deleted, as dir
    assertTrue(testFile.getParentFile().exists());  // not
}
```

Entrega 1

Eager Test

Refatorações

```
@Test
public void testFileCleanerExitWhenFinished_NoTrackAfter() {
    assertFalse(theInstance.exitWhenFinished);
    theInstance.exitWhenFinished();
    assertTrue(theInstance.exitWhenFinished);
    assertNull(theInstance.reaper);

    final String path = testFile.getPath();
    final Object marker = new Object();

    assertThrows(IllegalStateException.class, () -> theInstance.track(path, marker));
    assertTrue(theInstance.exitWhenFinished);
    assertNull(theInstance.reaper);
}
```

```
@Test
public void testFileCleanerExitWhenFinished_NoTrackAfter() {
    assertFalse(theInstance.exitWhenFinished);
    theInstance.exitWhenFinished();
    assertTrue(theInstance.exitWhenFinished);
    assertNull(theInstance.reaper);

    final String path = testFile.getPath();
    final Object marker = new Object();


    assertTrue(theInstance.exitWhenFinished);
    assertNull(theInstance.reaper);
}

@Test
public void testFileCleanerExitWhenFinished_NoTrackAfter2() {
    assertFalse(theInstance.exitWhenFinished);

    assertTrue(theInstance.exitWhenFinished);
    assertNull(theInstance.reaper);

    final String path = testFile.getPath();
    final Object marker = new Object();

    assertThrows(IllegalStateException.class, () -> theInstance.track(path, marker));
    assertTrue(theInstance.exitWhenFinished);
    assertNull(theInstance.reaper);
}
```

Entrega 1

Eager Test

Refatorações

```java
@Test
public void testFileCleanerDirectory_NullStrategy() throws Exception {
    TestUtils.createFile(testFile, 100);
    assertTrue(testFile.exists());
    assertTrue(tempDirFile.exists());

    Object obj = new Object();
    assertEquals(0, theInstance.getTrackCount());
    theInstance.track(tempDirFile, obj, null);
    assertEquals(1, theInstance.getTrackCount());

    obj = null;

    waitUntilTrackCount();

    assertEquals(0, theInstance.getTrackCount());
    assertTrue(testFile.exists());  // not deleted, as dir not empty
    assertTrue(testFile.getParentFile().exists());  // not deleted, as
}
```

```java
@Test
public void testFileCleanerDirectory_NullStrategy() throws Exception {
    TestUtils.createFile(testFile, 100);
    assertTrue(testFile.exists());
    assertTrue(tempDirFile.exists());

    Object obj = new Object();
    assertEquals(0, theInstance.getTrackCount());

    obj = null;

    waitUntilTrackCount();

    assertTrue(testFile.exists());  // not deleted, as dir not empty
    assertTrue(testFile.getParentFile().exists());  // not deleted, as dir
}

@Test
public void testFileCleanerDirectory_NullStrategy2() throws Exception {
    TestUtils.createFile(testFile, 100);
    assertTrue(testFile.exists());
    assertTrue(tempDirFile.exists());

    Object obj = new Object();

    assertEquals(1, theInstance.getTrackCount());

    obj = null;

    waitUntilTrackCount();


    assertTrue(testFile.exists());  // not deleted, as dir not empty
    assertTrue(testFile.getParentFile().exists());  // not deleted, as dir
}
```

Entrega 1

Eager Test

Refatorações

```java
@Test
public void testFileCleanerDirectory_NullStrategy() throws Exception {
    TestUtils.createFile(testFile, 100);
    assertTrue(testFile.exists());
    assertTrue(tempDirFile.exists());

    Object obj = new Object();
    assertEquals(0, theInstance.getTrackCount());
    theInstance.track(tempDirFile, obj, null);
    assertEquals(1, theInstance.getTrackCount());

    obj = null;

    waitUntilTrackCount();

    assertEquals(0, theInstance.getTrackCount());
    assertTrue(testFile.exists());  // not deleted, as dir not empty
    assertTrue(testFile.getParentFile().exists());  // not deleted, as
}
```

```java
@Test
public void testFileCleanerDirectory_NullStrategy3() throws Exception {
    TestUtils.createFile(testFile, 100);
    assertTrue(testFile.exists());
    assertTrue(tempDirFile.exists());

    Object obj = new Object();

    obj = null;

    waitUntilTrackCount();

    assertEquals(0, theInstance.getTrackCount());
    assertTrue(testFile.exists());  // not deleted, as dir not empty
    assertTrue(testFile.getParentFile().exists());  // not deleted, as di
}

@Test
public void testFileCleanerDirectory_NullStrategy4() throws Exception {
    TestUtils.createFile(testFile, 100);
    assertTrue(testFile.exists());
    assertTrue(tempDirFile.exists());

    Object obj = new Object();

    theInstance.track(tempDirFile, obj, null);

    obj = null;

    waitUntilTrackCount();

    assertTrue(testFile.exists());  // not deleted, as dir not empty
    assertTrue(testFile.getParentFile().exists());  // not deleted, as di
}
```

Entrega 1

Eager Test

Refatorações

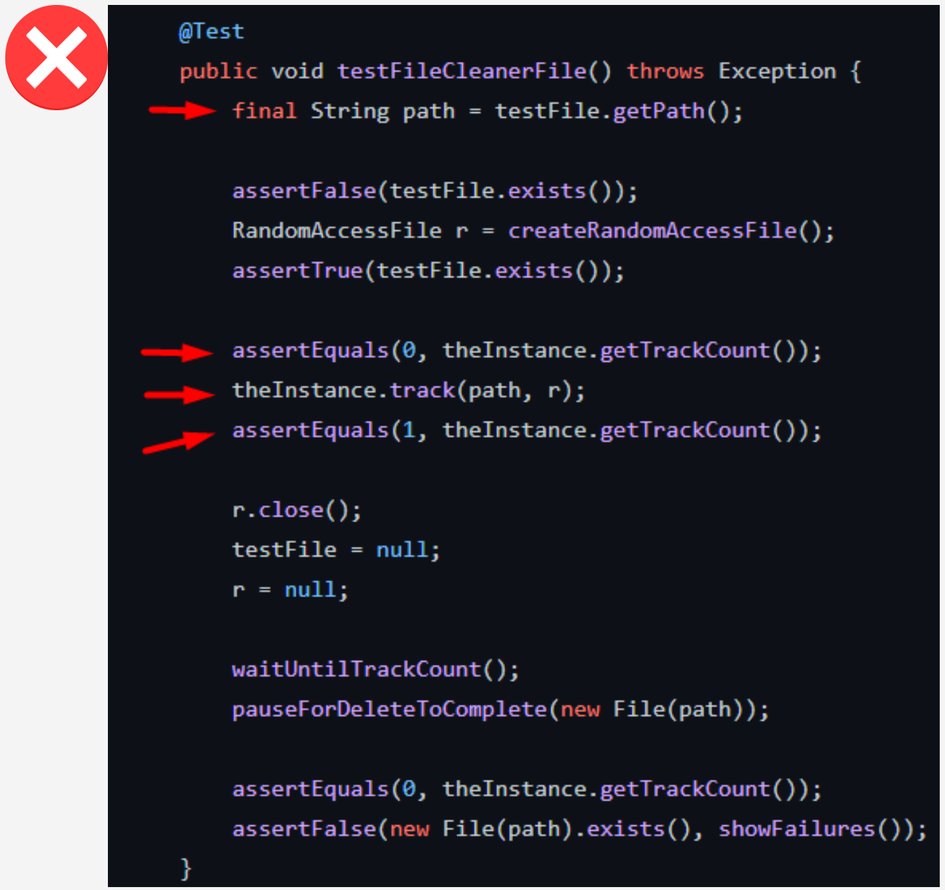
```java
@Test
public void testFileCleanerExitWhenFinishedFirst() throws Exception {
    assertFalse(theInstance.exitWhenFinished);
    theInstance.exitWhenFinished();
    assertTrue(theInstance.exitWhenFinished);
    assertNull(theInstance.reaper);

    waitUntilTrackCount();

    assertEquals(0, theInstance.getTrackCount());
    assertTrue(theInstance.exitWhenFinished);
    assertNull(theInstance.reaper);
}
```

```java
@Test
public void testFileCleanerExitWhenFinishedFirst() throws Exception {
    assertFalse(theInstance.exitWhenFinished);
    theInstance.exitWhenFinished();
    assertTrue(theInstance.exitWhenFinished);
    assertNull(theInstance.reaper);

    waitUntilTrackCount();

    assertTrue(theInstance.exitWhenFinished);
    assertNull(theInstance.reaper);
}

@Test
public void testFileCleanerExitWhenFinishedFirst_theInstancegetTrackCount()
    assertFalse(theInstance.exitWhenFinished);
    assertTrue(theInstance.exitWhenFinished);
    assertNull(theInstance.reaper);

    waitUntilTrackCount();

    assertEquals(0, theInstance.getTrackCount());
    assertTrue(theInstance.exitWhenFinished);
    assertNull(theInstance.reaper);
}
```

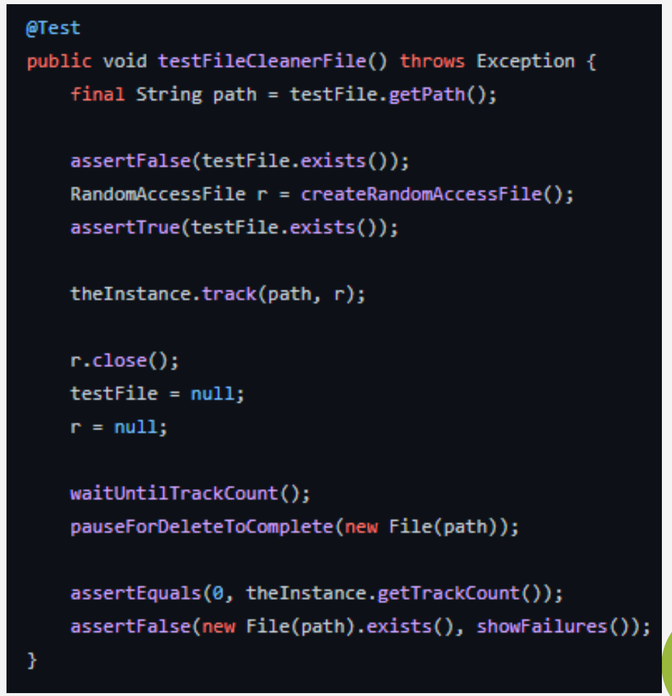
Entrega 1

Eager Test

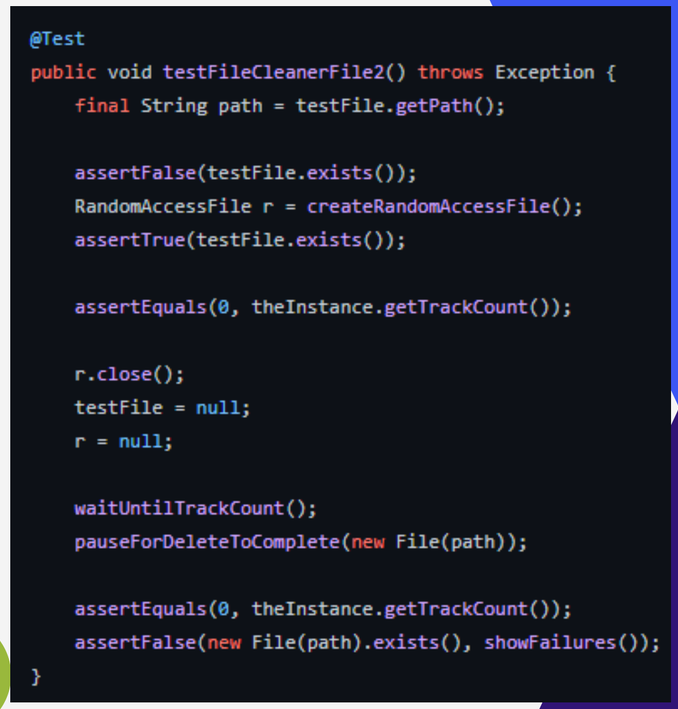
Refatorações



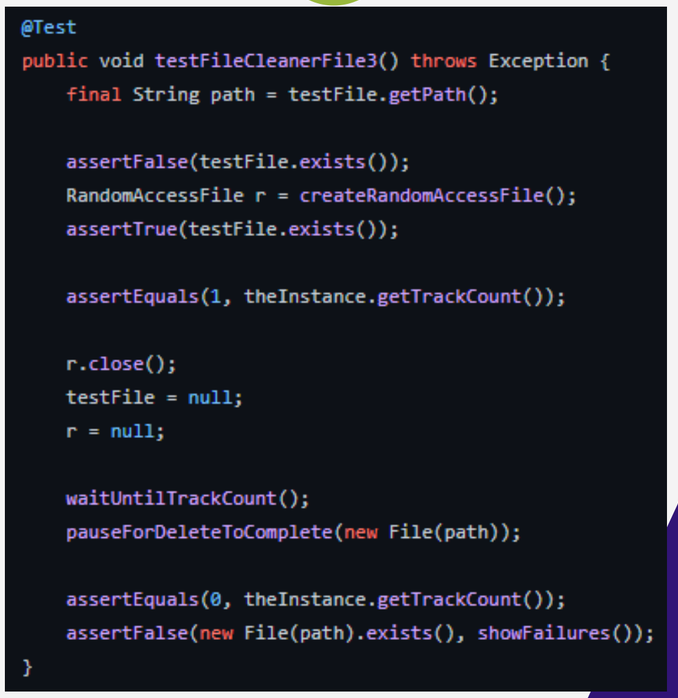
```java
@Test
public void testFileCleanerFile() throws Exception {
    final String path = testFile.getPath();

    assertFalse(testFile.exists());
    RandomAccessFile r = createRandomAccessFile();
    assertTrue(testFile.exists());

    assertEquals(0, theInstance.getTrackCount());
    theInstance.track(path, r);
    assertEquals(1, theInstance.getTrackCount());

    r.close();
    testFile = null;
    r = null;

    waitUntilTrackCount();
    pauseForDeleteToComplete(new File(path));

    assertEquals(0, theInstance.getTrackCount());
    assertFalse(new File(path).exists(), showFailures());
}
```



```java
@Test
public void testFileCleanerFile() throws Exception {
    final String path = testFile.getPath();

    assertFalse(testFile.exists());
    RandomAccessFile r = createRandomAccessFile();
    assertTrue(testFile.exists());

    theInstance.track(path, r);

    r.close();
    testFile = null;
    r = null;

    waitUntilTrackCount();
    pauseForDeleteToComplete(new File(path));

    assertEquals(0, theInstance.getTrackCount());
    assertFalse(new File(path).exists(), showFailures());
}
```



```java
@Test
public void testFileCleanerFile2() throws Exception {
    final String path = testFile.getPath();

    assertFalse(testFile.exists());
    RandomAccessFile r = createRandomAccessFile();
    assertTrue(testFile.exists());

    assertEquals(0, theInstance.getTrackCount());

    r.close();
    testFile = null;
    r = null;

    waitUntilTrackCount();
    pauseForDeleteToComplete(new File(path));

    assertEquals(0, theInstance.getTrackCount());
    assertFalse(new File(path).exists(), showFailures());
}
```



```java
@Test
public void testFileCleanerFile3() throws Exception {
    final String path = testFile.getPath();

    assertFalse(testFile.exists());
    RandomAccessFile r = createRandomAccessFile();
    assertTrue(testFile.exists());

    assertEquals(1, theInstance.getTrackCount());

    r.close();
    testFile = null;
    r = null;

    waitUntilTrackCount();
    pauseForDeleteToComplete(new File(path));

    assertEquals(0, theInstance.getTrackCount());
    assertFalse(new File(path).exists(), showFailures());
}
```

Entrega 1

Duplicate Assertion

Quantidade de test smells refatorados:

5 test smells refatorados;

Técnica de refatoração para remoção deste test smell:

Separar os asserts duplicados em classes de testes diferentes;

Quantidade de test smells removidos:

5 test smells removidos;

Entrega 1

Duplicate Assertion

Refatorações

```java
@Test
public void testDeleteForce() throws Exception {
    final File baseDir = temporaryFolder;
    final File subDir = new File(baseDir, "test");
    assertTrue(subDir.mkdir());
    final File subFile = new File(subDir, "a.txt");
    if (!subFile.getParentFile().exists()) {
        throw new IOException("Cannot create file " + subFile
                + " as the parent directory does not exist");
    }
    try (BufferedOutputStream output =
            new BufferedOutputStream(Files.newOutputStream(subFile.toPath()))) {
        TestUtils.generateTestData(output, 16);
    }

    assertTrue(subDir.exists());
    assertTrue(subFile.exists());
    // delete dir
    FileDeleteStrategy.FORCE.delete(subDir);
    assertFalse(subDir.exists());
    assertFalse(subFile.exists());
    // delete dir
    FileDeleteStrategy.FORCE.delete(subDir);  // no error
    assertFalse(subDir.exists());
}
```

```java
@Test
public void testDeleteForce() throws Exception {
    final File baseDir = temporaryFolder;
    final File subDir = new File(baseDir, "test");
    assertTrue(subDir.mkdir());
    final File subFile = new File(subDir, "a.txt");
    if (!subFile.getParentFile().exists()) {
        throw new IOException("Cannot create file " + subFile
                + " as the parent directory does not exist");
    }
    try (BufferedOutputStream output =
                new BufferedOutputStream(Files.newOutputStream(subFile.toPath()))) {
        TestUtils.generateTestData(output, 16);
    }

    assertTrue(subDir.exists());
    assertTrue(subFile.exists());

    // delete dir
    FileDeleteStrategy.FORCE.delete(subDir);
    assertFalse(subDir.exists());
    assertFalse(subFile.exists());
}
```

```java
@Test // testDeleteForce: Duplicate Assertion refactored
public void testDeleteForceNoError() throws Exception {
    final File baseDir = temporaryFolder;
    final File subDir = new File(baseDir, "test");
    assertTrue(subDir.mkdir());
    final File subFile = new File(subDir, "a.txt");
    if (!subFile.getParentFile().exists()) {
        throw new IOException("Cannot create file " + subFile
                + " as the parent directory does not exist");
    }
    try (BufferedOutputStream output =
                new BufferedOutputStream(Files.newOutputStream(subFile.toPath()))) {
        TestUtils.generateTestData(output, 16);
    }

    assertTrue(subDir.exists());
    assertTrue(subFile.exists());

    // delete dir
    FileDeleteStrategy.FORCE.delete(subDir);  // no error
    assertFalse(subDir.exists());
}
```

Entrega 1

Duplicate Assertion

Refatorações

```java
@Test
public void testDeleteNormal() throws Exception {
    final File baseDir = temporaryFolder;
    final File subDir = new File(baseDir, "test");
    assertTrue(subDir.mkdir());
    final File subFile = new File(subDir, "a.txt");
    if (!subFile.getParentFile().exists()) {
        throw new IOException("Cannot create file " + subFile
                + " as the parent directory does not exist");
    }
    try (BufferedOutputStream output =
            new BufferedOutputStream(Files.newOutputStream(subFile.toPath()))) {
        TestUtils.generateTestData(output, 16);
    }

    assertTrue(subDir.exists());
    assertTrue(subFile.exists());
    // delete dir
    assertThrows(IOException.class, () -> FileDeleteStrategy.NORMAL.delete(subDir));
    assertTrue(subDir.exists());
    assertTrue(subFile.exists());
    // delete file
    FileDeleteStrategy.NORMAL.delete(subFile);
    assertTrue(subDir.exists());
    assertFalse(subFile.exists());
    // delete dir
    FileDeleteStrategy.NORMAL.delete(subDir);
    assertFalse(subDir.exists());
    // delete dir
    FileDeleteStrategy.NORMAL.delete(subDir);  // no error
    assertFalse(subDir.exists());
}
```

Entrega 1

Duplicate Assertion

Refatorações

```java
@Test
public void testDeleteNormal() throws Exception {
    final File baseDir = temporaryFolder;
    final File subDir = new File(baseDir, "test");
    assertTrue(subDir.mkdir());
    final File subFile = new File(subDir, "a.txt");
    if (!subFile.getParentFile().exists()) {
        throw new IOException("Cannot create file " + subFile
                + " as the parent directory does not exist");
    }
    try (BufferedOutputStream output =
                 new BufferedOutputStream(Files.newOutputStream(subFile.toPath()))) {
        TestUtils.generateTestData(output, 16);
    }

    assertTrue(subDir.exists());
    assertTrue(subFile.exists());

}

@Test // testDeleteNormal: Duplicate Assertion refactored
public void testDeleteNormal1() throws Exception {
    final File baseDir = temporaryFolder;
    final File subDir = new File(baseDir, "test");
    assertTrue(subDir.mkdir());
    final File subFile = new File(subDir, "a.txt");
    if (!subFile.getParentFile().exists()) {
        throw new IOException("Cannot create file " + subFile
                + " as the parent directory does not exist");
    }
    try (BufferedOutputStream output =
                 new BufferedOutputStream(Files.newOutputStream(subFile.toPath()))) {
        TestUtils.generateTestData(output, 16);
    }

    // delete dir
    assertThrows(IOException.class, () -> FileDeleteStrategy.NORMAL.delete(subDir));
    assertTrue(subDir.exists());
    assertTrue(subFile.exists());
}
```

Entrega 1

Duplicate Assertion

Refatorações

```java
@Test // testDeleteNormal: Duplicate Assertion refactored
public void testDeleteNormal2() throws Exception {
    final File baseDir = temporaryFolder;
    final File subDir = new File(baseDir, "test");
    assertTrue(subDir.mkdir());
    final File subFile = new File(subDir, "a.txt");
    if (!subFile.getParentFile().exists()) {
        throw new IOException("Cannot create file " + subFile
                + " as the parent directory does not exist");
    }
    try (BufferedOutputStream output =
                 new BufferedOutputStream(Files.newOutputStream(subFile.toPath()))) {
        TestUtils.generateTestData(output, 16);
    }

    // delete file
    FileDeleteStrategy.NORMAL.delete(subFile);
    assertTrue(subDir.exists());
    assertFalse(subFile.exists());
}

@Test // testDeleteNormal: Duplicate Assertion refactored
public void testDeleteNormal3() throws Exception {
    final File baseDir = temporaryFolder;
    final File subDir = new File(baseDir, "test");
    assertTrue(subDir.mkdir());
    final File subFile = new File(subDir, "a.txt");
    if (!subFile.getParentFile().exists()) {
        throw new IOException("Cannot create file " + subFile
                + " as the parent directory does not exist");
    }
    try (BufferedOutputStream output =
                 new BufferedOutputStream(Files.newOutputStream(subFile.toPath()))) {
        TestUtils.generateTestData(output, 16);
    }

    // delete dir
    FileDeleteStrategy.NORMAL.delete(subFile);
    FileDeleteStrategy.NORMAL.delete(subDir);
    assertFalse(subDir.exists());
}
```

Entrega 1

Duplicate Assertion

Refatorações

```java
@Test // testDeleteNormal: Duplicate Assertion refactored
public void testDeleteNormal4() throws Exception {
    final File baseDir = temporaryFolder;
    final File subDir = new File(baseDir, "test");
    assertTrue(subDir.mkdir());
    final File subFile = new File(subDir, "a.txt");
    if (!subFile.getParentFile().exists()) {
        throw new IOException("Cannot create file " + subFile
                + " as the parent directory does not exist");
    }
    try (BufferedOutputStream output =
                 new BufferedOutputStream(Files.newOutputStream(subFile.toPath()))) {
        TestUtils.generateTestData(output, 16);
    }

    // delete dir
    FileDeleteStrategy.NORMAL.delete(subFile);
    FileDeleteStrategy.NORMAL.delete(subDir);  // no error
    assertFalse(subDir.exists());
}
```

Entrega 1

Duplicate Assertion

Refatorações

```java
@Test
public void testDeleteQuietlyNormal() throws Exception {
    final File baseDir = temporaryFolder;
    final File subDir = new File(baseDir, "test");
    assertTrue(subDir.mkdir());
    final File subFile = new File(subDir, "a.txt");
    if (!subFile.getParentFile().exists()) {
        throw new IOException("Cannot create file " + subFile
                + " as the parent directory does not exist");
    }
    try (BufferedOutputStream output =
            new BufferedOutputStream(Files.newOutputStream(subFile.toPath()))) {
        TestUtils.generateTestData(output, 16);
    }

    assertTrue(subDir.exists());
    assertTrue(subFile.exists());
    // delete dir
    assertFalse(FileDeleteStrategy.NORMAL.deleteQuietly(subDir));
    assertTrue(subDir.exists());
    assertTrue(subFile.exists());
    // delete file
    assertTrue(FileDeleteStrategy.NORMAL.deleteQuietly(subFile));
    assertTrue(subDir.exists());
    assertFalse(subFile.exists());
    // delete dir
    assertTrue(FileDeleteStrategy.NORMAL.deleteQuietly(subDir));
    assertFalse(subDir.exists());
    // delete dir
    assertTrue(FileDeleteStrategy.NORMAL.deleteQuietly(subDir));  // no error
    assertFalse(subDir.exists());
}
```

Entrega 1

Duplicate Assertion

Refatorações

```java
@Test
public void testDeleteQuietlyNormal() throws Exception {
    final File baseDir = temporaryFolder;
    final File subDir = new File(baseDir, "test");
    assertTrue(subDir.mkdir());
    final File subFile = new File(subDir, "a.txt");
    if (!subFile.getParentFile().exists()) {
        throw new IOException("Cannot create file " + subFile
                + " as the parent directory does not exist");
    }
    try (BufferedOutputStream output =
                new BufferedOutputStream(Files.newOutputStream(subFile.toPath()))) {
        TestUtils.generateTestData(output, 16);
    }

    assertTrue(subDir.exists());
    assertTrue(subFile.exists());

    // delete dir
    assertTrue(FileDeleteStrategy.NORMAL.deleteQuietly(subDir));
    assertFalse(subDir.exists());
    // delete dir
    assertTrue(FileDeleteStrategy.NORMAL.deleteQuietly(subDir));  // no error
    assertFalse(subDir.exists());
}
```

```java
@Test
public void testDeleteQuietlyNormal1() throws Exception {
    final File baseDir = temporaryFolder;
    final File subDir = new File(baseDir, "test");
    assertTrue(subDir.mkdir());
    final File subFile = new File(subDir, "a.txt");
    if (!subFile.getParentFile().exists()) {
        throw new IOException("Cannot create file " + subFile
                + " as the parent directory does not exist");
    }
    try (BufferedOutputStream output =
                new BufferedOutputStream(Files.newOutputStream(subFile.toPath()))) {
        TestUtils.generateTestData(output, 16);
    }

    // delete dir
    assertFalse(FileDeleteStrategy.NORMAL.deleteQuietly(subDir));
    assertTrue(subDir.exists());
    assertTrue(subFile.exists());

    // delete dir
    assertTrue(FileDeleteStrategy.NORMAL.deleteQuietly(subDir));
    assertFalse(subDir.exists());
    // delete dir
    assertTrue(FileDeleteStrategy.NORMAL.deleteQuietly(subDir));  // no error
    assertFalse(subDir.exists());
}
```

Entrega 1

Duplicate Assertion

Refatorações

```java
@Test
public void testDeleteQuietlyNormal2() throws Exception {
    final File baseDir = temporaryFolder;
    final File subDir = new File(baseDir, "test");
    assertTrue(subDir.mkdir());
    final File subFile = new File(subDir, "a.txt");
    if (!subFile.getParentFile().exists()) {
        throw new IOException("Cannot create file " + subFile
                + " as the parent directory does not exist");
    }
    try (BufferedOutputStream output =
                new BufferedOutputStream(Files.newOutputStream(subFile.toPath()))) {
        TestUtils.generateTestData(output, 16);
    }

    // delete file
    assertTrue(FileDeleteStrategy.NORMAL.deleteQuietly(subFile));
    assertTrue(subDir.exists());
    assertFalse(subFile.exists());

    // delete dir
    assertTrue(FileDeleteStrategy.NORMAL.deleteQuietly(subDir));
    assertFalse(subDir.exists());
    // delete dir
    assertTrue(FileDeleteStrategy.NORMAL.deleteQuietly(subDir));  // no error
    assertFalse(subDir.exists());
}
```

Consolidação dos diários

- **Dificuldades encontradas na remoção dos test smells**

  Assertion Roulette

  Entender o contexto do que precisava ser feito.

  Eager Test

  Separar os objetos e garantir as boas práticas e convenções.

  Duplicate Assertion

  Entender como separar sem prejudicar o teste.

Consolidação dos diários

- **O quão prejudicial os test smells são para o sistema?**

  Assertion Roulette

  Afeta a manutenção dos testes.

  Eager Test

  Afeta a manutenção e compreensão dos testes.

  Duplicate Assertion

  Alguns módulos podem não ser testados.

Entrega 2

Ignored Test

Quantidade de test smells refatorados:

5 test smells refatorados;

Técnica de refatoração para remoção deste test smell:

Verificar se a classe possui o "@Ignore" e deletá-las,
pois estavam vazias;

Quantidade de test smells removidos:

5 test smells removidos;

Entrega 2

Ignored Test

Refatorações

```
@Override
@Test
@Ignore("not supported by the BasicParser")
public void testLongWithoutEqualSingleDash() throws Exception {
}
```

Entrega 2

Ignored Test

Refatorações

```
@Override
@Test
@Ignore("not supported by the BasicParser")
public void testMissingArgWithBursting() throws Exception {
}
```

Entrega 2

Ignored Test

Refatorações

```
@Override
@Test
@Ignore("not supported by the BasicParser (CLI-184)")
public void testNegativeOption() throws Exception {
}
```

Entrega 2

Ignored Test

Refatorações

```
@Override
@Test
@Ignore("not supported by the BasicParser")
public void testPartialLongOptionSingleDash() throws Exception {
}
```

Entrega 2

Ignored Test

Refatorações

```
@Override
@Test
@Ignore("not supported by the BasicParser")
public void testPropertiesOption1() throws Exception {
}
```

Entrega 2

Magic Number Test

Quantidade de test smells refatorados:

5 test smells refatorados;

Técnica de refatoração para remoção deste test smell:

Substituir os literais numéricos por variáveis que forncem uma descrição do que é esperado no assert;

Quantidade de test smells removidos:

5 test smells removidos;

Entrega 2

Magic Number Test

Refatorações

```
 assertEquals(0, ((PerUserPoolDataSource) ds).getNumIdle(user),
         "Should be no idle connections in the pool");
```

```
int noNumIdleExpected = 0;
assertEquals(noNumIdleExpected, ((PerUserPoolDataSource) ds).getNumIdle(user),
        "Should be no idle connections in the pool");
```

Entrega 2

Magic Number Test

Refatorações

```
assertEquals(1, ((PerUserPoolDataSource) ds).getNumIdle(user),
        "Should be one idle connection in the pool");
```

```
int yesNumIdleExpected = 1;
assertEquals(yesNumIdleExpected, ((PerUserPoolDataSource) ds).getNumIdle(user),
        "Should be one idle connection in the pool");
```

Entrega 2

Magic Number Test

Refatorações

```
assertEquals(1, ((PerUserPoolDataSource) ds).getNumIdle(user),
        "Should be one idle connection in the pool");
```

```
int conNumIdleExpected = 1;
assertEquals(conNumIdleExpected, ((PerUserPoolDataSource) ds).getNumIdle(user),
        "Should be one idle connection in the pool");
```

Entrega 2

Magic Number Test

Refatorações

```
assertEquals(0, tds.getNumActive());
assertEquals(0, tds.getNumActive("u1"));
```

```
int numActiveExpected = 0;
assertEquals(numActiveExpected, tds.getNumActive());

int u1NumActiveExpected = 0;
assertEquals(u1NumActiveExpected, tds.getNumActive("u1"));
```

Entrega 2

Lazy Test

Quantidade de test smells refatorados:

5 test smells refatorados;

Técnica de refatoração para remoção deste test smell:

Observação das classes de teste que chamam o mesmo método do objeto, e juntando estes métodos dentro de uma classe de teste.

Quantidade de test smells removidos:

5 test smells removidos;

Entrega 2

Lazy Test

Refatorações

- `filter.correct(z);` Linhas 146 e 238
- `Assert.assertTrue(Precision.compareTo(filter.getErrorCovariance()[1][1], 0.1d, 1e-6) < 0);` Linhas 246 e 155
- `assertMatrixEquals(Q.getData(), filter.getErrorCovariance());` Linhas 211 e 119
- `filter.predict(u);` Linhas 224 e 132
- `Assert.assertEquals(1, filter.getMeasurementDimension());` Linhas 208 e 116
- `Assert.assertEquals(2, filter.getStateDimension());` Linhas 209 e 117

Entrega 2

Lazy Test

Refatorações

União dos métodos testConstantAcceleration com testConstant

```java
@Test
public void testConstant() {
    // simulates a simple process with a constant state and no control input

    double constantValue = 10d;
    double measurementNoise = 0.1d;
    double processNoise = 1e-5d;

    // A = [ 1 ]
    RealMatrix A = new Array2DRowRealMatrix(new double[] { 1d });
    // no control input
    RealMatrix B = null;
    // H = [ 1 ]
    RealMatrix H = new Array2DRowRealMatrix(new double[] { 1d });
    // x = [ 10 ]
    RealVector x = new ArrayRealVector(new double[] { constantValue });
    // Q = [ 1e-5 ]
    RealMatrix Q = new Array2DRowRealMatrix(new double[] { processNoise });
    // R = [ 0.1 ]
    RealMatrix R = new Array2DRowRealMatrix(new double[] { measurementNoise });

    ProcessModel pm
        = new DefaultProcessModel(A, B, Q,
                                  new ArrayRealVector(new double[] { constantValue }), null);
    MeasurementModel mm = new DefaultMeasurementModel(H, R);
    KalmanFilter filter = new KalmanFilter(pm, mm);

    Assert.assertEquals(1, filter.getMeasurementDimension());
    Assert.assertEquals(1, filter.getStateDimension());

    assertMatrixEquals(Q.getData(), filter.getErrorCovariance());

    // check the initial state
    double[] expectedInitialState = new double[] { constantValue };
    assertVectorEquals(expectedInitialState, filter.getStateEstimation());

    RealVector pNoise = new ArrayRealVector(1);
    RealVector mNoise = new ArrayRealVector(1);

    final ContinuousSampler rand = createGaussianSampler(0, 1);

    // iterate 60 steps
    for (int i = 0; i < 60; i++) {
        filter.predict();

        // Simulate the process
        pNoise.setEntry(0, processNoise * rand.sample());

        // x = A * x + p_noise
        x = A.operate(x).add(pNoise);

        // Simulate the measurement
        mNoise.setEntry(0, measurementNoise * rand.sample());

        // z = H * x + m_noise
        RealVector z = H.operate(x).add(mNoise);

        filter.correct(z);

        // state estimate shouldn't be larger than measurement noise
        double diff = JdkMath.abs(constantValue - filter.getStateEstimation()[0]);
        // System.out.println(diff);
        Assert.assertTrue(Precision.compareTo(diff, measurementNoise, 1e-6) < 0);
    }

    // error covariance should be already very low (< 0.02)
    Assert.assertTrue(Precision.compareTo(filter.getErrorCovariance()[0][0],
                                          0.02d, 1e-6) < 0);

    //  **** union methods ****

    // simulates a vehicle, accelerating at a constant rate (0.1 m/s)

    // discrete time interval
    double dt_acceleration = 0.1d;
    // position measurement noise (meter)
    double measurementNoise_acceleration = 10d;
    // acceleration noise (meter/sec^2)
    double accelNoise_acceleration = 0.2d;

    // A = [ 1 dt ]
    //     [ 0  1 ]
    RealMatrix A_acceleration = new Array2DRowRealMatrix(new double[][] { { 1, dt_acceleration }, { 0, 1 } });

    // B = [ dt^2/2 ]
    //     [ dt     ]
    RealMatrix B_acceleration = new Array2DRowRealMatrix(
        ProcessModel pm_acceleration = new DefaultProcessModel(A_acceleration, B_acceleration, Q_acceleration, x_acceleration, P0_acceleration);
        MeasurementModel mm_acceleration = new DefaultMeasurementModel(H_acceleration, R_acceleration);
        KalmanFilter filter_acceleration = new KalmanFilter(pm_acceleration, mm_acceleration);

        Assert.assertEquals(1, filter_acceleration.getMeasurementDimension());
        Assert.assertEquals(2, filter_acceleration.getStateDimension());

        assertMatrixEquals(P0_acceleration.getData(), filter_acceleration.getErrorCovariance());

        // check the initial state
        double[] expectedInitialState_acceleration = new double[] { 0.0, 0.0 };
        assertVectorEquals(expectedInitialState_acceleration, filter_acceleration.getStateEstimation());

        final ContinuousSampler rand_acceleration = createGaussianSampler(0, 1);

        RealVector tmpPNoise_acceleration = new ArrayRealVector(
                new double[] { JdkMath.pow(dt_acceleration, 2d) / 2d, dt_acceleration });

        // iterate 60 steps
        for (int i = 0; i < 60; i++) {
            filter_acceleration.predict(u_acceleration);

            // Simulate the process
            RealVector pNoise_acceleration = tmpPNoise_acceleration.mapMultiply(accelNoise_acceleration * rand_acceleration.sample());

            // x = A * x + B * u + pNoise
            x_acceleration = A_acceleration.operate(x_acceleration).add(B_acceleration.operate(u_acceleration)).add(pNoise_acceleration);

            // Simulate the measurement
            double mNoise_acceleration = measurementNoise_acceleration * rand_acceleration.sample();

            // z = H * x + m_noise
            RealVector z_acceleration = H_acceleration.operate(x_acceleration).mapAdd(mNoise_acceleration);

            filter_acceleration.correct(z_acceleration);

            // state estimate shouldn't be larger than the measurement noise
            double diff_acceleration = JdkMath.abs(x_acceleration.getEntry(0) - filter_acceleration.getStateEstimation()[0]);
            Assert.assertTrue(Precision.compareTo(diff_acceleration, measurementNoise_acceleration, 1e-6) < 0);
        }

        // error covariance of the velocity should be already very low (< 0.1)
        Assert.assertTrue(Precision.compareTo(filter_acceleration.getErrorCovariance()[1][1],
                0.1d, 1e-6) < 0);
}
```

Consolidação dos diários

Dificuldades encontradas na remoção dos test smells

Ignored Test

Refatoração tranquila, pois é basicamente remoção.

Magic Number Test

Entender o que era esperado para nomear corretamente a variável.

Lazy Test

Entender como juntar classes de teste sem perder a corretude do teste, e conseguir acoplar isso tudo.

Consolidação dos diários

- **O quão prejudicial os test smells são para o sistema?**

  Ignored Test

  Pode gerar sobrecarga no tempo de compilação e dificuldade na compreensão.

  Magic Number Test

  Dificulta a compreensão e pode gerar problemas evolutivos dos testes.

  Lazy Test

  Gera dificuldade em manter com constância a manutenção dos testes.

Entrega 3

Constructor Inicialization

Quantidade de test smells refatorados:

5 test smells refatorados;

Técnica de refatoração para remoção deste test smell:

Adicionar a assinatura @Before nas classes construtoras.

Quantidade de test smells removidos:

5 test smells removidos;

Entrega 3

Constructor Inicialization

Refatorações

```
public BKRegistrationNameResolverTest() {
    super(0);
    this.resolverProvider = new BKRegistrationNameResolverProvider();
}
```

```
@Before
public BKRegistrationNameResolverTest() {
    super(0);
    this.resolverProvider = new BKRegistrationNameResolverProvider();
}
```

Entrega 3

Constructor Inicialization

Refatorações

```
public DefaultStorageContainerControllerTest() {
    this.controller = new DefaultStorageContainerController();
    this.clusterMetadata = ClusterMetadata.newBuilder()
        .setNumStorageContainers(NUM_STORAGE_CONTAINERS)
        .build();
```

```
@Before
public DefaultStorageContainerControllerTest() {
    this.controller = new DefaultStorageContainerController();
    this.clusterMetadata = ClusterMetadata.newBuilder()
        .setNumStorageContainers(NUM_STORAGE_CONTAINERS)
        .build();
```

Entrega 3

Constructor Inicialization

Refatorações

```
public ListUnderReplicatedCommandTest() {
    super(3, 0);
}
```

```
@Before
public ListUnderReplicatedCommandTest() {
    super(3, 0);
}
```

Entrega 3

Constructor Inicialization

Refatorações

```
public QueryAutoRecoveryStatusCommandTest() {
    super(3, 0);
}
```

```
@Before
public QueryAutoRecoveryStatusCommandTest() {
    super(3, 0);
}
```

Entrega 3

Constructor Inicialization

Refatorações

```
public TriggerAuditCommandTest() {
    super(3, 0);
}
```

```
@Before
public TriggerAuditCommandTest() {
    super(3, 0);
}
```

Entrega 3

Print Statement

Quantidade de test smells refatorados:

5 test smells refatorados;

Técnica de refatoração para remoção deste test smell:

Retirar as impressões da classe de teste, e caso haja chamadas de métodos, coloca-los fora da impressão.

Quantidade de test smells removidos:

5 test smells removidos;

Entrega 3

Print Statement

Refatorações

```
futures.forEach(
        runnableFuture -> {
            NettyPartitionRequestClient client;
            try {
                client = runnableFuture.get();
                assertThat(client).isNotNull();
            } catch (Exception e) {
                System.out.println(e.getMessage());
                fail();
            }
        });
```

```
futures.forEach(
        runnableFuture -> {
            NettyPartitionRequestClient client;
            try {
                client = runnableFuture.get();
                assertThat(client).isNotNull();
            } catch (Exception e) {
                e.getMessage();
                fail();
            }
        });
```

Entrega 3

Print Statement

Refatorações

```
System.out.println("valid window trigger");
RowIterator<BinaryRowData> iter = grouping.buildTriggerWindowElementsIterator();
TimeWindow window = grouping.getTriggerWindow();
List<Long> buffer = new ArrayList<>();
while (iter.advanceNext()) {
    buffer.add(iter.getRow().getLong(0));
}
```

```
assertThat(window).isEqualTo(TimeWindow.of(0, 8L));
assertThat(buffer).isEmpty();


System.out.println("try invalid window trigger");
grouping.buildTriggerWindowElementsIterator();
```

Entrega 3

Print Statement

Refatorações

```
// should have retried sending a join group request already
assertFalse(client.hasPendingResponses());
assertEquals(1, client.inFlightRequestCount());

System.out.println(client.requests());
```

```
// should have retried sending a join group request already
assertFalse(client.hasPendingResponses());
assertEquals(1, client.inFlightRequestCount());

client.requests();
```

Entrega 3

Print Statement

Refatorações

```
private void testInvalidRule(String rules) {
    try {
        System.out.println(SslPrincipalMapper.fromRules(rules));
        fail("should have thrown IllegalArgumentException");
    } catch (IllegalArgumentException e) {
    }
}
```

```
private void testInvalidRule(String rules) {
    try {
        SslPrincipalMapper.fromRules(rules);
        fail("should have thrown IllegalArgumentException");
    } catch (IllegalArgumentException e) {
    }
}
```

Entrega 3

Unknown Test

Quantidade de test smells refatorados:

5 test smells refatorados;

Técnica de refatoração para remoção deste test smell:

Inserir assertivas na classe de teste;

Quantidade de test smells removidos:

5 test smells removidos;

Entrega 3

Unknown Test

Refatorações

```
@Test
void renderHomePage() throws Exception
{
        executeTest(SimplePage.class, "SimplePageExpectedResult.html");
}
```

```
@Test
void renderHomePage() throws Exception
{
        executeTest(SimplePage.class, "SimplePageExpectedResult.html");
        assertEquals("SimplePageExpectedResult.html", executeTest.filename);
}
```

Entrega 3

Unknown Test

Refatorações

```
@Test
void renderHomePage_3() throws Exception
{
        executeTest(SimplePage_3.class, "SimplePageExpectedResult_3.html");
}
```

```
@Test
void renderHomePage_3() throws Exception
{
        executeTest(SimplePage_3.class, "SimplePageExpectedResult_3.html")
        assertEquals("SimplePageExpectedResult_3.html", executeTest.filename);
}
```

Entrega 3

Unknown Test

Refatorações

```
@Test
void renderHomePage_7() throws Exception
{
        tester.getApplication().getResourceSettings().setThrowExceptionOnMissingResource(false);
        // This is for issue https://issues.apache.org/jira/browse/WICKET-590
        executeTest(SimplePage_7.class, "SimplePageExpectedResult_7.html");
}
```

```
@Test
void renderHomePage_7() throws Exception
{
        tester.getApplication().getResourceSettings().setThrowExceptionOnMissingResource(false);
        // This is for issue https://issues.apache.org/jira/browse/WICKET-590
        executeTest(SimplePage_7.class, "SimplePageExpectedResult_7.html");
        assertEquals("SimplePageExpectedResult_7.html", executeTest.filename);
}
```

Entrega 3

Unknown Test

Refatorações

```
@Test
void renderHomePage_9() throws Exception
{
        executeTest(SimplePage_9.class, "SimplePageExpectedResult_9.html");
}
```

```
@Test
void renderHomePage_9() throws Exception
{
        executeTest(SimplePage_9.class, "SimplePageExpectedResult_9.html");
        assertEquals("SimplePageExpectedResult_9.html", executeTest.filename);
}
```

Entrega 3

Unknown Test

Refatorações

```
@Test
void renderHomePage_11() throws Exception
{
        executeTest(SimplePage_11.class, "SimplePageExpectedResult_11.html");
}
```

```
@Test
void renderHomePage_11() throws Exception
{
        executeTest(SimplePage_11.class, "SimplePageExpectedResult_11.html");
        assertEquals("SimplePageExpectedResult_11.html", executeTest.filename);
}
```

Consolidação dos diários

Dificuldades encontradas na remoção dos test smells

Constructor Inicialization

Refatoração tranquila, pois é basicamente a adição da assinatura @Before no construtor.

Print Statement

Entender se o método chamado na impressão poderia ser removido ou mantido.

Unknown Test

Entender o contexto do teste para conseguir fazer uma assertiva adequada àquele cenário.

Consolidação dos diários

- **O quão prejudicial os test smells são para o sistema?**

  Constructor Inicialization

  Pode gerar resultados inesperados por métodos construtores serem executados apenas uma vez;

  Print Statement

  Impressões são desnecessárias em métodos de teste e aumentam o tempo de execução do teste;

  Unknown Test

  O JUnit aprova o método de teste e prejudica a compreensão do teste.

Conclusão

- **Lições aprendidas com o trabalho**

  - **Conhecimento em uma nova área;**
  - **Prática em refatoração de variados test smells;**
  - **Fortalecimento na construção de testes unitários mais concisos.**

Referências

RAIDE: Uma abordagem semi-automatizada para identificação e refatoração de test smells. Dissertação de Mestrado: Railana dos Santos Santana. Publicado em: julho de 2020. Disponível em: https://repositorio.ufba.br/bitstream/ri/33621/1/Dissertacao_RailanaSantana_VersaoFinal.pdf

Software Unit Test Smells. Disponível em: https://testsmells.org/

[UFC] Qualidade de Software. Materiais de aula e vídeos. Humberto Damasceno e profa. Carla Bezerra.

OBRIGADO!