Tests Smells

Gabriel Maia Gondim - 478943
Wilhelm Steins - 495961

Code smells refatorados

Entrega 1:

- Assertion roulette;
- Duplicate assertion;
- Eager test.

Entrega 2:

- Ignored test;
- Magic number test;
- Lazy test.

Entrega 3:

- Constructor initialization;
- Print statement;
- Unknown test.

Assertion roulette

Ocorre quando há várias asserções não documentadas num mesmo caso de teste.
Correção: Documentar asserções.

Exemplo:

```
@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)));
}
```

Duplicate assertion

Ocorre quando a mesma asserção se repete na mesma função de teste.
Correção: Separar os casos de testes com a mesma asserção em funções diferentes.

Exemplo:

```
@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'));
```

```
@Test
public void testContains_CharEx1() {
    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_CharEx2() {
    CharRange range = CharRange.isIn('c', 'd');
    assertFalse(range.contains('b'));
    assertTrue(range.contains('c'));
    assertTrue(range.contains('d'));
    assertFalse(range.contains('e'));
}
```

Eager test

Ocorre quando o teste executa vários testes do objeto de produção.
Correção: Separar cada uso de uma função do objeto de produção em uma função de teste diferente.

Exemplo:

```java
@Test
public void testDefaultValueOfUseClassName() {
    assertTrue((new StandardToStringStyle()).isUseClassName());
}

@Test
public void testDefaultValueOfUseFieldNames() {
    assertTrue((new StandardToStringStyle()).isUseFieldNames());
}
```

```java
private final StandardToStringStyle defaultStyle = new StandardToStringStyle();

...

@Test
public void testDefaultValueOfUseClassName() {
    assertTrue(defaultStyle.isUseClassName());
}

@Test
public void testDefaultValueOfUseFieldNames() {
    assertTrue(defaultStyle.isUseFieldNames());
}
```

Entrega 1

Dificuldades:
Os eager tests foram os únicos mais complicados de resolver (porque tem que quebrar a função de teste corretamente usando apenas uma função em cada função de teste). Os outros foram fáceis de resolver.

Refatorações:

- Assertion roulette: Adicionar documentação na asserção;
- Duplicate assertion: Extrair método;
- Eager tests: Extrair variável.

Quão prejudicial esses test smells são (de 0 a 10):

- Assertion roulette: 6;
- Duplicate assertion: 4;
- Eager tests: 3.

Entrega 1

Dificuldades:
Os Eager Tests foram os que deram mais trabalho para refatorar, os outros dois eram mais facilmente visíveis.

Refatorações:

- Assertion roulette: Adicionar string de documentação na asserção;
- Duplicate assertion: Extrair método com muitas asserções parecidas em mais métodos;
- Eager tests: Extrair variável deixando o método menos complexo.

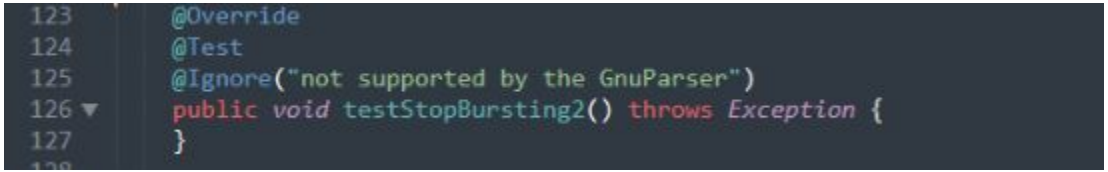
Quão prejudicial esses test smells são (de 0 a 10):

- Assertion roulette: 5, complica na identificação do que aquele teste está testando;
- Duplicate assertion: 5, método fica com muitas asserções parecidas;
- Eager tests: 7, o método fica muito complexo;

Ignored test

Ocorre quando há um teste com a anotação @Ignore.
Correção: Se o teste for vazio, só remover o teste. Se ele tiver algo testado, retirar o ignore e corrigir o teste ou remover o teste.

Exemplo:



Magic number test

Ocorre quando é feita uma asserção com um número escolhido “magicamente”, sem ter explicação do número em si.
Correção: Extrair número para variável com nome que explique o que o número significa.

Exemplo:

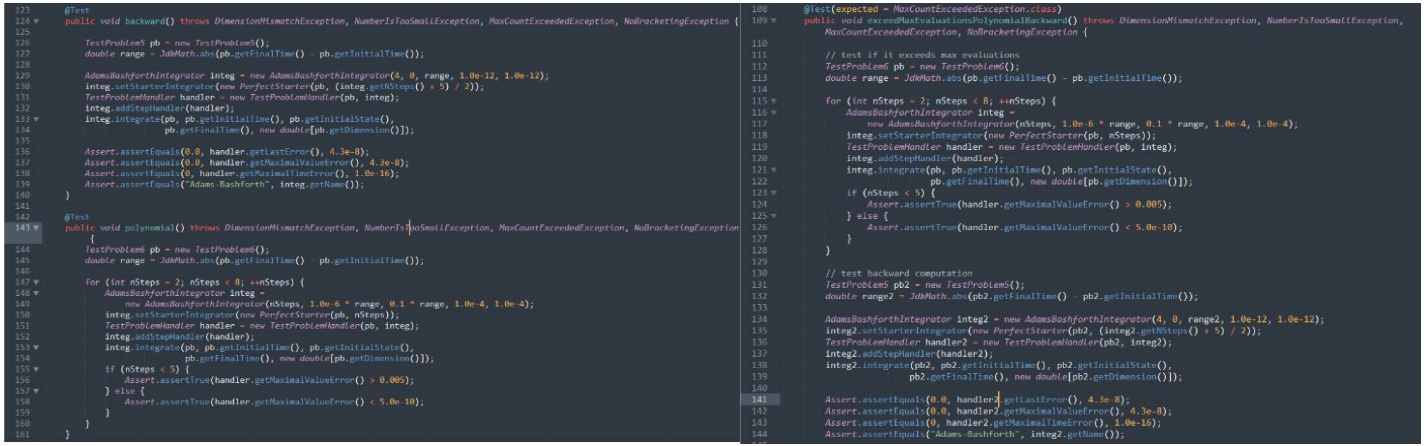
```
68          final ThreadPoolExecutor evictionExecutor = (ThreadPoolExecutor) evictorExecutorField.get(null);
69          assertEquals(2, evictionExecutor.getQueue().size()); // Reaper plus one eviction task
70          assertEquals(1, EvictionTimer.getNumTasks());
71
```

```
70          int expectedQueueSize = 2;
71          int expectedNumberTasks = 1;
72
73          assertEquals(expectedQueueSize, evictionExecutor.getQueue().size()); // Reaper plus one eviction task
74          assertEquals(expectedNumberTasks, EvictionTimer.getNumTasks());
75
```

Lazy test

Ocorre quando a mesma função do objeto de produção é chamada em várias funções de teste diferentes.
Correção: Colocar todos os usos da função em questão em uma mesma função de teste.

Exemplo:



```
@Test
public void backward() throws DimensionMismatchException, NumberIsTooSmallException, MaxCountExceededException, NoBracketingException {

    TestProblem5 pb = new TestProblem5();
    double range = JdkMath.abs(pb.getFinalTime() - pb.getInitialTime());

    AdamsBashforthIntegrator integ = new AdamsBashforthIntegrator(4, 0, range, 1.0e-12, 1.0e-12);
    integ.setStarterIntegrator(new PerfectStarter(pb, (integ.getNSteps() + 5) / 2));
    TestProblemHandler handler = new TestProblemHandler(pb, integ);
    integ.addStepHandler(handler);
    integ.integrate(pb, pb.getInitialTime(), pb.getInitialState(),
                    pb.getFinalTime(), new double[pb.getDimension()]);

    Assert.assertEquals(0.0, handler.getLastError(), 4.3e-8);
    Assert.assertEquals(0.0, handler.getMaximalValueError(), 4.3e-8);
    Assert.assertEquals(0, handler.getMaximalTimeError(), 1.0e-16);
    Assert.assertEquals("Adams-Bashforth", integ.getName());
}

@Test
public void polynomial() throws DimensionMismatchException, NumberIsTooSmallException, MaxCountExceededException, NoBracketingException
    {
    TestProblem6 pb = new TestProblem6();
    double range = JdkMath.abs(pb.getFinalTime() - pb.getInitialTime());

    for (int nSteps = 2; nSteps < 8; ++nSteps) {
        AdamsBashforthIntegrator integ =
            new AdamsBashforthIntegrator(nSteps, 1.0e-6 * range, 0.1 * range, 1.0e-4, 1.0e-4);
        integ.setStarterIntegrator(new PerfectStarter(pb, nSteps));
        TestProblemHandler handler = new TestProblemHandler(pb, integ);
        integ.addStepHandler(handler);
        integ.integrate(pb, pb.getInitialTime(), pb.getInitialState(),
                        pb.getFinalTime(), new double[pb.getDimension()]);
        if (nSteps < 5) {
            Assert.assertTrue(handler.getMaximalValueError() > 0.005);
        } else {
            Assert.assertTrue(handler.getMaximalValueError() < 5.0e-10);
        }
    }
}
```

```
@Test(expected = MaxCountExceededException.class)
public void exceedMaxEvaluationsPolynomialBackward() throws DimensionMismatchException, NumberIsTooSmallException,
    MaxCountExceededException, NoBracketingException {

    // test if it exceeds max evaluations
    TestProblem6 pb = new TestProblem6();
    double range = JdkMath.abs(pb.getFinalTime() - pb.getInitialTime());

    for (int nSteps = 2; nSteps < 8; ++nSteps) {
        AdamsBashforthIntegrator integ =
            new AdamsBashforthIntegrator(nSteps, 1.0e-6 * range, 0.1 * range, 1.0e-4, 1.0e-4);
        integ.setStarterIntegrator(new PerfectStarter(pb, nSteps));
        TestProblemHandler handler = new TestProblemHandler(pb, integ);
        integ.addStepHandler(handler);
        integ.integrate(pb, pb.getInitialTime(), pb.getInitialState(),
                        pb.getFinalTime(), new double[pb.getDimension()]);
        if (nSteps < 5) {
            Assert.assertTrue(handler.getMaximalValueError() > 0.005);
        } else {
            Assert.assertTrue(handler.getMaximalValueError() < 5.0e-10);
        }
    }

    // test backward computation
    TestProblem5 pb2 = new TestProblem5();
    double range2 = JdkMath.abs(pb2.getFinalTime() - pb2.getInitialTime());

    AdamsBashforthIntegrator integ2 = new AdamsBashforthIntegrator(4, 0, range2, 1.0e-12, 1.0e-12);
    integ2.setStarterIntegrator(new PerfectStarter(pb2, (integ2.getNSteps() + 5) / 2));
    TestProblemHandler handler2 = new TestProblemHandler(pb2, integ2);
    integ2.addStepHandler(handler2);
    integ2.integrate(pb2, pb2.getInitialTime(), pb2.getInitialState(),
                     pb2.getFinalTime(), new double[pb2.getDimension()]);

    Assert.assertEquals(0.0, handler2.getLastError(), 4.3e-8);
    Assert.assertEquals(0.0, handler2.getMaximalValueError(), 4.3e-8);
    Assert.assertEquals(0, handler2.getMaximalTimeError(), 1.0e-16);
    Assert.assertEquals("Adams-Bashforth", integ2.getName());
```

Entrega 2

Dificuldades:
Todos os testes foram simples de resolver, especialmente o ignored test, que foi só apagar as funções.

Refatorações:

- Ignored test: Remover método vazio;
- Magic number test: Extrair variável;
- Lazy test: Método inline, unindo funções.

Quão prejudicial esses test smells são (de 0 a 10):

- Ignored test: 4;
- Magic number test: 7;
- Lazy test: 2.

Entrega 2

Dificuldades:
Todos os testes foram simples de resolver, especialmente o ignored test, que foi só apagar as funções.

Refatorações:

- Ignored test: Remover método vazio;
- Magic number test: Criar variável para o número;
- Lazy test: Método inline, unindo funções.

Quão prejudicial esses test smells são (de 0 a 10):

- Ignored test: 3, o contador de testes vai considerar o ignored test como sucesso;
- Magic number test: 2, número inserido no código sem explicação, confusão ao rever o método;
- Lazy test: 2, métodos funcionavam mesmo separados, questão de organização de código;

Constructor initialization

Ocorre quando uma classe de teste inicializa propriedades na construtora no lugar da função setup.
Correção: Colocar inicializações da construtora na função setup.

Exemplo:

```java
public abstract class BookieCommandTestBase extends CommandTestBase {

    protected final int numJournalDirs;
    protected final int numLedgerDirs;

    protected BookieCommandTestBase(int numJournalDirs, int numLedgerDirs) {
        this.numJournalDirs = numJournalDirs;
        this.numLedgerDirs = numLedgerDirs;
    }
```

```java
public abstract class BookieCommandTestBase extends CommandTestBase {

    protected int numJournalDirs;
    protected int numLedgerDirs;

    protected BookieCommandTestBase() {
    }

    protected BookieCommandTestBase(int numJournalDirs, int numLedgerDirs) {
        this.numJournalDirs = numJournalDirs;
        this.numLedgerDirs = numLedgerDirs;
    }
```

```java
public class InitCommandTest extends BookieCommandTestBase {

    public InitCommandTest() {
        super(3, 0);
    }

    @Override
    public void setup() throws Exception {
        super.setup();

        mockServerConfigurationConstruction();
        final MockedStatic<BookKeeperAdmin> bookKeeperAdminMockedStatic = mockStatic(BookKeeperAdmin.class);
        bookKeeperAdminMockedStatic.when(() -> BookKeeperAdmin.initNewCluster(any(ServerConfiguration.class)))
                .thenReturn(true);
    }
```

```java
public class InitCommandTest extends BookieCommandTestBase {
    @Override
    public void setup() throws Exception {
        numJournalDirs = 3;
        numLedgerDirs = 0;

        super.setup();

        mockServerConfigurationConstruction();
        final MockedStatic<BookKeeperAdmin> bookKeeperAdminMockedStatic = mockStatic(BookKeeperAdmin.class);
        bookKeeperAdminMockedStatic.when(() -> BookKeeperAdmin.initNewCluster(any(ServerConfiguration.class)))
                .thenReturn(true);
    }
```

Print statement

Ocorre quando há alguma instrução de print no teste.
Correção: Remover instruções de print do teste.

Exemplo:

```
126     @Test
127     public void installUninstallCommand() throws Exception {
128         System.out.println(executeCommand("feature:install -v -r wrapper", new RolePrincipal("admin")));
129         assertFeatureInstalled("wrapper");
130         System.out.println(executeCommand("feature:uninstall -r wrapper", new RolePrincipal("admin")));
131         assertFeatureNotInstalled("wrapper");
132     }
```

```
124     @Test
125     public void installUninstallCommand() throws Exception {
126         executeCommand("feature:install -v -r wrapper", new RolePrincipal("admin"));
127         assertFeatureInstalled("wrapper");
128         executeCommand("feature:uninstall -r wrapper", new RolePrincipal("admin"));
129         assertFeatureNotInstalled("wrapper");
130     }
```

Unknown test

Ocorre quando há um teste sem nenhuma asserção.
Correção: Adicionar asserção ao teste.

Exemplo:

```
144     @Test
145     public void testServerHostnameVerificationWithHostnameVerificationDisabled() throws Exception {
146         ZKTrustManager zkTrustManager = new ZKTrustManager(mockX509ExtendedTrustManager, false, false);
147
148         X509Certificate[] certificateChain = createSelfSignedCertifcateChain(IP_ADDRESS, HOSTNAME);
149         zkTrustManager.checkServerTrusted(certificateChain, null, mockSocket);
150
151         verify(mockInetAddress, times(0)).getHostAddress();
152         verify(mockInetAddress, times(0)).getHostName();
153
154         verify(mockX509ExtendedTrustManager, times(1)).checkServerTrusted(certificateChain, null, mockSocket);
155     }
```

```
145     @Test
146     public void testServerHostnameVerificationWithHostnameVerificationDisabled() throws Exception {
147         ZKTrustManager zkTrustManager = new ZKTrustManager(mockX509ExtendedTrustManager, false, false);
148
149         X509Certificate[] certificateChain = createSelfSignedCertifcateChain(IP_ADDRESS, HOSTNAME);
150         assertDoesNotThrow(() -> zkTrustManager.checkServerTrusted(certificateChain, null, mockSocket));
151
152         verify(mockInetAddress, times(0)).getHostAddress();
153         verify(mockInetAddress, times(0)).getHostName();
154
155         verify(mockX509ExtendedTrustManager, times(1)).checkServerTrusted(certificateChain, null, mockSocket);
156     }
```

Entrega 3

Dificuldades:
No de constructor initialization, as classes derivavam de uma classe base onde ficava a construtora. Então a classe base também teve que ser mudada. Os outros dois testes foram bem simples de resolver.

Refatorações:

- Constructor initialization: Colapsar construtora;
- Print statement: Remover instruções de print;
- Unknown test: Adicionar asserção.

Quão prejudicial esses test smells são (de 0 a 10):

- Constructor initialization: 5;
- Print statement: 6;
- Unknown test: 9 no geral, 6 nesse caso.

Entrega 3

Dificuldades:
No de constructor initialization, as classes derivam de uma classe base onde ficava a construtora. Então a classe base também teve que ser mudada. Os outros dois testes foram bem simples de resolver.

Refatorações:

- Constructor initialization: Colapsar construtora;
- Print statement: Remover instruções de print;
- Unknown test: Adicionar asserção.

Quão prejudicial esses test smells são (de 0 a 10):

- Constructor initialization: 7, acaba gerando uma confusão nos contructors das classes;
- Print statement: 3, não prejudica o funcionamento do test em si;
- Unknown test: 10, é necessário um assert ou então o test fica inútil, pois pode verificar se é sucesso por outro método não convencional, compreensível pela ferramenta de teste;

Conclusão