Experimento de Refatoração de Test Smells Parte 3

Nome: Gabriel Maia Gondim - 478943

Projeto: https://github.com/gabrielmaia2/TrabalhoQS3

Eu estou atualmente trabalhando na refatoração dos test smells seguintes:

5 test smells do tipo constructor initialization nos arquivos \bookkeeper\tools\ledger\src\test\java\org\apache\bookkeeper\tools\cli\commands\bookies\InitCommandTest.java, \bookkeeper\tools\ledger\src\test\java\org\apache\bookkeeper\tools\cli\commands\bookies\InstanceIdCommandTest.java, \bookkeeper\tools\ledger\src\test\java\org\apache\bookkeeper\tools\cli\commands\bookies\MetaFormatCommandTest.java, \bookkeeper\tools\ledger\src\test\java\org\apache\bookkeeper\tools\cli\commands\bookies\NukeExistingClusterCommandTest.java e \bookkeeper\tools\ledger\src\test\java\org\apache\bookkeeper\tools\cli\commands\bookies\RecoverCommandTest.java

Código original e refatorado de \bookkeeper\tools\ledger\src\test\java\org\apache\bookkeeper\tools\cli\helpers\BookieCommandTestBase.java:

```
27 ▼ public abstract class BookieCommandTestBase extends CommandTestBase {
28
29       protected final int numJournalDirs;
30       protected final int numLedgerDirs;
31
32 ▼     protected BookieCommandTestBase(int numJournalDirs, int numLedgerDirs) {
33           this.numJournalDirs = numJournalDirs;
34           this.numLedgerDirs = numLedgerDirs;
35       }
```

```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;
    }
```

Código original e refatorado de \bookkeeper\tools\ledger\src\test\java\org\apache\bookkeeper\tools\cli\commands\bookies\InitCommandTest.java:

```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);
    }
```

Código original e refatorado de \bookkeeper\tools\ledger\src\test\java\org\apache\bookkeeper\tools\cli\commands\bookies\InstanceIdCommandTest.java:

```java
public class InstanceIdCommandTest extends BookieCommandTestBase {

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

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

        final RegistrationManager manager = mock(RegistrationManager.class);
        mockMetadataDriversWithRegistrationManager(manager);
        when(manager.getClusterInstanceId()).thenReturn("");
    }
```

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

        super.setup();

        final RegistrationManager manager = mock(RegistrationManager.class);
        mockMetadataDriversWithRegistrationManager(manager);
        when(manager.getClusterInstanceId()).thenReturn("");
    }
```

Código original e refatorado de \bookkeeper\tools\ledger\src\test\java\org\apache\bookkeeper\tools\cli\commands\bookies\MetaFormatCommandTest.java:

```java
public class MetaFormatCommandTest extends BookieCommandTestBase {

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

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

        final MockedStatic<BookKeeperAdmin> bookKeeperAdminMockedStatic = mockStatic(BookKeeperAdmin.class);
        bookKeeperAdminMockedStatic.when(() -> BookKeeperAdmin.format(any(ServerConfiguration.class),
                        anyBoolean(), anyBoolean())).thenReturn(true);

    }
```

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

        super.setup();

        final MockedStatic<BookKeeperAdmin> bookKeeperAdminMockedStatic = mockStatic(BookKeeperAdmin.class);
        bookKeeperAdminMockedStatic.when(() -> BookKeeperAdmin.format(any(ServerConfiguration.class),
                        anyBoolean(), anyBoolean())).thenReturn(true);

    }
```

Código original e refatorado de \bookkeeper\tools\ledger\src\test\java\org\apache\bookkeeper\tools\cli\commands\bookies\NukeExistingClusterCommandTest.java:

```java
public class NukeExistingClusterCommandTest extends BookieCommandTestBase {

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

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

    }
```

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

        super.setup();

    }
```

Código original e refatorado de \bookkeeper\tools\ledger\src\test\java\org\apache\bookkeeper\tools\cli\commands\bookies\RecoverCommandTest.java:

```java
    public RecoverCommandTest() {
        super(3, 0);
    }

    @Override
    public void setup() throws Exception {
        super.setup();
        mockServerConfigurationConstruction();
        mockClientConfigurationConstruction();
        ledgerMetadata = mock(LedgerMetadata.class);
        registrationManager = mock(RegistrationManager.class);
        cookieVersioned = mock(Versioned.class);


        mockBkQuery();
        mockDeleteCookie();
        mockDeleteCookies();
    }
```

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

    super.setup();

    mockServerConfigurationConstruction();
    mockClientConfigurationConstruction();
    ledgerMetadata = mock(LedgerMetadata.class);
    registrationManager = mock(RegistrationManager.class);
    cookieVersioned = mock(Versioned.class);


    mockBkQuery();
    mockDeleteCookie();
    mockDeleteCookies();
}
```

5 Test smells do tipo print statement no arquivo \karaf\itests\test\src\test\java\org\apache\karaf\itests\FeatureTest.java

Código original:

```java
@Test
public void listCommand() throws Exception {
    String featureListOutput = executeCommand("feature:list");
    System.out.println(featureListOutput);
    assertFalse(featureListOutput.isEmpty());
    featureListOutput = executeCommand("feature:list -i");
    System.out.println(featureListOutput);
    assertFalse(featureListOutput.isEmpty());
}
```

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

@Test
public void upgradeUninstallCommand() throws Exception {
    System.out.println(executeAlias("feature:upgrade -v -r wrapper", new RolePrincipal("admin")));
    assertFeatureInstalled("wrapper");
    System.out.println(executeCommand("feature:uninstall -r wrapper", new RolePrincipal("admin")));
    assertFeatureNotInstalled("wrapper");
}
```

Código refatorado:

```java
    @Test
    public void listCommand() throws Exception {
        String featureListOutput = executeCommand("feature:list");
        assertFalse(featureListOutput.isEmpty());
        featureListOutput = executeCommand("feature:list -i");
        assertFalse(featureListOutput.isEmpty());
    }
```

```java
    @Test
    public void installUninstallCommand() throws Exception {
        executeCommand("feature:install -v -r wrapper", new RolePrincipal("admin"));
        assertFeatureInstalled("wrapper");
        executeCommand("feature:uninstall -r wrapper", new RolePrincipal("admin"));
        assertFeatureNotInstalled("wrapper");
    }

    @Test
    public void upgradeUninstallCommand() throws Exception {
        executeAlias("feature:upgrade -v -r wrapper", new RolePrincipal("admin"));
        assertFeatureInstalled("wrapper");
        executeCommand("feature:uninstall -r wrapper", new RolePrincipal("admin"));
        assertFeatureNotInstalled("wrapper");
    }
```

5 Test smells do tipo unknown test no arquivo \zookeeper\zookeeper-server\src\test\java\org\apache\zookeeper\common\ZKTrustManagerTest.java

Código original:

```java
    @Test
    public void testServerHostnameVerificationWithHostnameVerificationDisabled() throws Exception {
        ZKTrustManager zkTrustManager = new ZKTrustManager(mockX509ExtendedTrustManager, false, false);

        X509Certificate[] certificateChain = createSelfSignedCertifcateChain(IP_ADDRESS, HOSTNAME);
        zkTrustManager.checkServerTrusted(certificateChain, null, mockSocket);

        verify(mockInetAddress, times(0)).getHostAddress();
        verify(mockInetAddress, times(0)).getHostName();

        verify(mockX509ExtendedTrustManager, times(1)).checkServerTrusted(certificateChain, null, mockSocket);
    }

    @Test
    public void testServerHostnameVerificationWithHostnameVerificationDisabledAndClientHostnameVerificationEnabled() throws Exception {
        ZKTrustManager zkTrustManager = new ZKTrustManager(mockX509ExtendedTrustManager, false, true);

        X509Certificate[] certificateChain = createSelfSignedCertifcateChain(IP_ADDRESS, HOSTNAME);
        zkTrustManager.checkServerTrusted(certificateChain, null, mockSocket);

        verify(mockInetAddress, times(0)).getHostAddress();
        verify(mockInetAddress, times(0)).getHostName();

        verify(mockX509ExtendedTrustManager, times(1)).checkServerTrusted(certificateChain, null, mockSocket);
    }
```

```
170     @Test
171     public void testServerHostnameVerificationWithIPAddress() throws Exception {
172         ZKTrustManager zkTrustManager = new ZKTrustManager(mockX509ExtendedTrustManager, true, false);
173
174         X509Certificate[] certificateChain = createSelfSignedCertifcateChain(IP_ADDRESS, null);
175         zkTrustManager.checkServerTrusted(certificateChain, null, mockSocket);
176
177         verify(mockInetAddress, times(1)).getHostAddress();
178         verify(mockInetAddress, times(0)).getHostName();
179
180         verify(mockX509ExtendedTrustManager, times(1)).checkServerTrusted(certificateChain, null, mockSocket);
181     }
182
183     @Test
184     public void testServerHostnameVerificationWithHostname() throws Exception {
185         ZKTrustManager zkTrustManager = new ZKTrustManager(mockX509ExtendedTrustManager, true, false);
186
187         X509Certificate[] certificateChain = createSelfSignedCertifcateChain(null, HOSTNAME);
188         zkTrustManager.checkServerTrusted(certificateChain, null, mockSocket);
189
190         verify(mockInetAddress, times(1)).getHostAddress();
191         verify(mockInetAddress, times(1)).getHostName();
192
193         verify(mockX509ExtendedTrustManager, times(1)).checkServerTrusted(certificateChain, null, mockSocket);
194     }
195
```

```
196     @Test
197     public void testClientHostnameVerificationWithHostnameVerificationDisabled() throws Exception {
198         ZKTrustManager zkTrustManager = new ZKTrustManager(mockX509ExtendedTrustManager, false, true);
199
200         X509Certificate[] certificateChain = createSelfSignedCertifcateChain(null, HOSTNAME);
201         zkTrustManager.checkClientTrusted(certificateChain, null, mockSocket);
202
203         verify(mockInetAddress, times(1)).getHostAddress();
204         verify(mockInetAddress, times(1)).getHostName();
205
206         verify(mockX509ExtendedTrustManager, times(1)).checkClientTrusted(certificateChain, null, mockSocket);
207     }
```

Código refatorado:

```
21  import static org.junit.jupiter.api.Assertions.assertDoesNotThrow;
```

```
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     }
157
158     @Test
159     public void testServerHostnameVerificationWithHostnameVerificationDisabledAndClientHostnameVerificationEnabled() throws Exception {
160         ZKTrustManager zkTrustManager = new ZKTrustManager(mockX509ExtendedTrustManager, false, true);
161
162         X509Certificate[] certificateChain = createSelfSignedCertifcateChain(IP_ADDRESS, HOSTNAME);
163         assertDoesNotThrow(() -> zkTrustManager.checkServerTrusted(certificateChain, null, mockSocket));
164
165         verify(mockInetAddress, times(0)).getHostAddress();
166         verify(mockInetAddress, times(0)).getHostName();
167
168         verify(mockX509ExtendedTrustManager, times(1)).checkServerTrusted(certificateChain, null, mockSocket);
169     }
170
```

```java
    @Test
    public void testServerHostnameVerificationWithIPAddress() throws Exception {
        ZKTrustManager zkTrustManager = new ZKTrustManager(mockX509ExtendedTrustManager, true, false);

        X509Certificate[] certificateChain = createSelfSignedCertifcateChain(IP_ADDRESS, null);
        assertDoesNotThrow(() -> zkTrustManager.checkServerTrusted(certificateChain, null, mockSocket));

        verify(mockInetAddress, times(1)).getHostAddress();
        verify(mockInetAddress, times(0)).getHostName();

        verify(mockX509ExtendedTrustManager, times(1)).checkServerTrusted(certificateChain, null, mockSocket);
    }

    @Test
    public void testServerHostnameVerificationWithHostname() throws Exception {
        ZKTrustManager zkTrustManager = new ZKTrustManager(mockX509ExtendedTrustManager, true, false);

        X509Certificate[] certificateChain = createSelfSignedCertifcateChain(null, HOSTNAME);
        assertDoesNotThrow(() -> zkTrustManager.checkServerTrusted(certificateChain, null, mockSocket));

        verify(mockInetAddress, times(1)).getHostAddress();
        verify(mockInetAddress, times(1)).getHostName();

        verify(mockX509ExtendedTrustManager, times(1)).checkServerTrusted(certificateChain, null, mockSocket);
    }

```

```java
    @Test
    public void testClientHostnameVerificationWithHostnameVerificationDisabled() throws Exception {
        ZKTrustManager zkTrustManager = new ZKTrustManager(mockX509ExtendedTrustManager, false, true);

        X509Certificate[] certificateChain = createSelfSignedCertifcateChain(null, HOSTNAME);
        assertDoesNotThrow(() -> zkTrustManager.checkClientTrusted(certificateChain, null, mockSocket));

        verify(mockInetAddress, times(1)).getHostAddress();
        verify(mockInetAddress, times(1)).getHostName();

        verify(mockX509ExtendedTrustManager, times(1)).checkClientTrusted(certificateChain, null, mockSocket);
    }
```

Minhas principais dificuldades ao remover essas anormalidades foram:

Para os constructor initialization test smells, no código as cinco classes de teste estendiam uma classe base e usavam uma construtora dela que definia dois campos protegidos da classe, então eu tive que mexer não só nas classes de teste, mas também encontrar e modificar a classe base destas.

Os test smells do tipo print statement foram bem fáceis de resolver, simplesmente removi os println do código, deixando só o restante do código que era relevante para o teste.

Os do tipo unknown test foram também bastante simples, bastava adicionar um assert verificando que a exceção não era disparada.

Eu estou usando as seguintes técnicas de refatoração para remover test smells:

Constructor initialization: Colapsar construtora, movendo os campos definidos na construtora para as funções de setup().

Print statement: Remover chamadas a System.out.println() nos testes.

Unknown test: No caso a função testada no lugar de retornar um booleano dizendo se a verificação deu sucesso ou falha, ela disparava uma exceção CertificateException caso

desse falha. Coloquei um assert para verificar se a exceção não era jogada no caso (para garantir que a operação sempre tem sucesso).

De 0 a 10, quão prejudicial é esse test smell para o sistema? Por que?

Constructor initialization: 5.
Nesse caso não é tão prejudicial pelo fato de as construtoras estarem simplesmente definindo variáveis numéricas, mas nesse caso e no geral ainda continua sendo um antipadrão e confuso.

Print statement: 6.
Não prejudica a funcionalidade do teste, mas pode confundir o desenvolvedor e também aumentar o tempo de teste, já que chamadas a funções como println são bastante custosas.

Unknown test: 9 no geral, 6 nesse caso.
No geral, um teste que não faz nenhuma asserção é um teste inútil e confuso, pois ele pode estar verificando se dá sucesso de alguma forma não convencional, que não seria entendida por uma ferramenta de teste na execução dos testes. Nesse caso, já haviam algumas asserções sendo feitas com as funções verify do mockito, que verificam se um método foi chamado ou não, faltando portanto apenas garantir que o certificado verificado na função chamada era válido.