Experimento de Refatoração de Test Smells

Nome: Wilhelm de Sousa Steins - 495961

Projeto: Repositório

Eu estou atualmente trabalhando na refatoração dos test smells seguintes:

Código antes:

```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;
    }
```

Código refatorado:

```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 antes:

```
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);
    }
```

Código refatorado:

```
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 antes:

```
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("");
    }
```

Código refatorado:

```
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 antes:

```
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);

    }
```

Código refatorado:

```
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 antes:

```
public class NukeExistingClusterCommandTest extends BookieCommandTestBase {

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

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

    }
```

Código refatorado:

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

        super.setup();

    }
```

Código antes:

```
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();
}
```

Código refatorado:

```
@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();
}
```

Código antes:

```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");
}
```

Código antes:

```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);
}
```

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

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

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

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

Código refatorado:

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

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

    X509Certificate[] certificateChain = createSelfSignedCertifcateChain(IP_ADDRESS, HOSTNAME);
    assertDoesNotThrow(() -> 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);
    assertDoesNotThrow(() -> 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 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);
}
```

```
@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: Constructor initialization test smell, principal dificuldade foi ter que refatorar tanto a classe principal quanto a que estava sendo utilizada como base. Print statement test smells, eram mais fáceis pois era só remover o print do código. Unknown test smells, precisava apenas adicionar um assert para verificação se era ativada a exceção.
Eu estou usando as seguintes técnicas de refatoração para remover test smells: Constructor initialization, colapsar construtora Print statement, remover os prints da classe Unknown, usar o assert para garantir o sucesso da operação
De 0 a 10, quão prejudicial é esse test smell para o sistema? Por que? Constructor initialization, 7 gera muita confusão nos contructors das classes. Print statement, 3 não necessariamente prejudica o funcionamento do teste em si. Unknown, 10 pois os testes são necessários da existência de um assert, caso não o teste fica inútil.