Tesztelés jegyzőkönyv

# Tesztelés során felhasznált eszközök

Az egységtesztelést a JUnit egységtesztelő keretrendszer segítségével végeztem. A JUnit az xUnit egységtesztelő keretrendszerek családjába tartozik. A nevében a „J” utal arra, hogy Java programozási nyelven íródott forráskódok teszteléséhez készült, és a teszteket szintén Java programozási nyelven kell implementálni. A JUnit támogatja az iteratív és inkrementális tesztelési és fejlesztési irányelveket. Ilyen például a teszt vezérelt fejlesztést (TDD – Test Driven Development), amely azt jelenti, hogy az új funkció lefejlesztése előtt, először a kódot tesztelő osztályokat implementáljuk, és csak ezután fejlesztjük le a tényleges funkciót. A JUnit-nak már több verziója is elérhető. A tesztelést a legújabb verzióval (JUnit 5) valósítottam meg.

# Tesztelt osztályok

org.jkiss.dbeaver.ui.data.dialogs.TextViewDialog: Az általam implementált JSON Beautifier feature megvalósítása a TextViewDialog osztályban található. Ez az osztály felel a CTRL + Shift lenyomására felugró ablakban található különböző editorokért, ezért egyértelmű volt, hogy a tesztelése szükséges. Azonban a tesztek tervezése/írása közben kiderült (és részben már a feature fejlesztése közben és érezhető volt), hogy a kód nem megfelelő minősége miatt elég nehézkes tesztelni az osztályt. Sajnálatos módon a mockolással sem sikerült előbbre haladni, így végül az általam írt két metódust teszteltem le az osztályban.

org.jkiss.utils.xml.XMLUtils: Az XMLUtils osztály, ahogy nevéből is rá lehet jönni, egy utility osztály, melyben különböző XML-lel kapcsolatos statikus metódusok, függvények találhatóak. Azért választottam tesztelésre az osztályt, mivel egyrészt jellegéből adódóan könnyebben tesztelhető, másrészt, mivel utility osztályról beszélünk, ezért több helyen is használhatják, ezért fontos, hogy a működése megfelelő legyen.

# Létrehozott tesztcsomag bemutatása

org.jkiss.dbeaver.ui.data.dialogs.TextViewDialogTest: A TextViewDialog osztályon belüli isJSON() és parseToJson() metódusok tesztelését végzi. Az egyik metódus a JSON String validálásért felel (isJSON()), a másik pedig a String JSONStringgé (indentált, formázott) alakításért felel. Próbálkoztam az osztály többi metódusának a tesztelésével is, de ez nem volt sikeres, mivel egyrészt sok volt a privát metódus, másrészt a TextViewDialog-ot nem lehetett megfelelően példányosítani (Ahogy feljebb írtam, mockolással sem lehetett megoldani). Tesztelve lett a metódusok működése negatív és pozitív input esetén, illetve a parseToJson() indentálása/formázása is tesztelve lett.

org.jkiss.utils.xml.XMLUtilsTest: Az XMLUtils osztályon belüli parseDocument(), createDocument(), getChildElement(), getChildElementBody() és getElementBody() metódusok tesztelését végzi. A parseDocument() XML dokumentumot próbál létrehozni a beolvasott (valószínű) XML dokumentumból/fájlból. A tesztelése során Stringet használtam mint átalakítandó forrást (egy kis előfeldolgozással, hogy át tudjam adni a metódusnak). Ellenőriztem, hogy ténylegesen létrehozta-e a Document objektumot, illetve, hogy tartalmazta-e az általam megadott elemet, ami bizonyítja, hogy sikeres volt-e a parseolás. Emellett teszteltem, hiba esetén (nem valid XML) dob-e hibát. A createDocument() létrehoz egy új Document objektumot (org.w3c.dom.Document). A getChildElement() segítségével lekérhetjük egy XML elem adott nevű gyerekét. Teszteltem pozitív inputra (az elemnek van adott nevű gyereke is) és negatív inputra is (az elemnek nincs adott nevű gyereke). A getElementBody() vissza adja az elem értékét/bodyját (<test>dbeaver</test> esetén a dbeaver-t). Itt is az előzőhez hasonló megközelítést alkalmaztam, vagyis teszteltem egy pozitív és egy negatív inputra. A getChildElementBody(), ahogy a nevében és látszik, vissza adja az XML elem megadott nevű gyerekének a bodyját. Itt egy pozitív és két negatív tesztesetet csináltam. (Egyik esetben az elemnek nem volt megadott nevű gyereke, míg a másik esetben a gyereknek nem volt bodyja).

# Létrehozott tesztek pontos megadása

org.jkiss.dbeaver.ui.data.dialogs.TextViewDialogTest.java:

1. testIsJSONWithInvalidString():

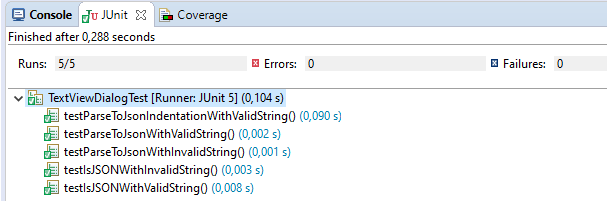
* Arrange: 24-26. sorok
* Act és Assert: 40-42. sorok

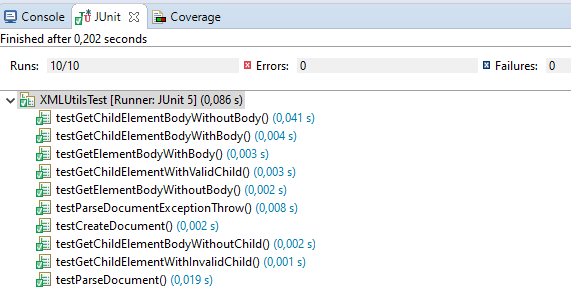
1. testIsJSONWithValidString():
   * Arrange: 28-31. sorok
   * Act és Assert: 47-50. sorok
2. testParseToJsonWithInvalidString():
   * Arrange: 24-26. sorok
   * Act és Assert: 55-57. sorok
3. testParseToJsonWithValidString():
   * Arrange: 29-31. sorok
   * Act és Assert: 63-65. sorok
4. testParseToJsonIndentationWithValidString():
   * Arrange: 28, 33-34. sorok
   * Act és Assert: 71-73. sorok

org.jkiss.utils.xml.XMLUtilsTest.java

1. testCreateDocument():
   * Arrange és Act: 45. sor
   * Assert: 46. sor
2. testParseDocument():
   * Arrange: 35, 55. sorok
   * Act: 57, 59. sorok
   * Assert: 58, 60. sorok
3. testParseDocumentExceptionThrow():
   * Arrange: 36, 70. sorok
   * Act és Assert: 71. sor
4. testGetChildElementWithValidChild():
   * Arrange: 37, 76-80. sorok
   * Act: 81. sor
   * Assert: 82-83. sorok
5. testGetChildElementWithInvalidChild():
   * Arrange: 35, 92-96. sorok
   * Act: 97. sor
   * Assert: 98. sor
6. testGetElementBodyWithBody():
   * Arrange: 38, 108-112. sorok
   * Act: 113. sor
   * Assert: 114. sor
7. testGetElementBodyWithoutBody ():
   * Arrange: 35, 124-128. sorok
   * Act: 129. sor
   * Assert: 130. sor
8. testGetChildElementBodyWithBody ():
   * Arrange: 39, 140-144. sorok
   * Act: 145. sor
   * Assert: 146. sor
9. testGetChildElementBodyWithoutBody ():
   * Arrange: 37, 156-160. sorok
   * Act: 161. sor
   * Assert: 162. sor
10. testGetChildElementBodyWithoutChild ():
    * Arrange: 35, 172-176. sorok
    * Act: 177. sor
    * Assert: 178. sor

# Végrehajtott tesztek sikerességének megállapítása

org.jkiss.dbeaver.ui.data.dialogs.TextViewDialogTest.java: Mind az 5 teszt sikeres volt.

org.jkiss.utils.xml.XMLUtilsTest.java: Mind az 10 teszt sikeres volt.

Lefedettség jelentés

# Lefedettség mérés során felhasznált eszközök

A kód lefedettséget SonarQoube segítségével mértem. A SonarQoube egy open source szoftver, amely a forráskód minőségének ellenőrzésére és követésére szolgál. A SonarQoube képes a Java programozási nyelven íródott forráskódok feldolgozására. Különböző grafikonokat készít a forráskód minőségi kritériumairól. Ilyen kritérium lehet például a duplikált kód mennyisége, a kód szabványok betartása, unit tesztek mennyisége és az általuk lefedett forráskód, a forráskód komplexitása, esetleges hibák száma, dokumentáltság.

# Lefedettségi érték indoklása

Én az *org.jkiss.dbeaver.ui.data.dialogs* csomag lefedettségét mértem, mivel ez volt az egyik csomag, amelyben dolgoztam. A lefedettsége sajnos elég alacsony, 1.6%

*org.jkiss.dbeaver.ui.data.dialogs* csomag méret metrikái:

1. Lines of Code: 1,749
2. Lines: 2,145
3. Statements: 763
4. Functions: 138
5. Classes: 20
6. Files: 5
7. Comment Lines: 39
8. Comments (%): 2.2%

*org.jkiss.dbeaver.ui.data.dialogs* csomag tényleges lefedettsége:

1. Coverage: 1.6%
2. Lines to Cover: 902
3. Uncovered Lines: 887
4. Lines Coverage: 1.7%

Ez az érték azért lehet probléma, mert így nem lehet biztosak program ezen részének megfelelő, elvárt működésében. Az elvárt érték legalább 70-80 % körül kéne lenni.

# Erősen és kevésbe lefedett kódelemek bemutatása

Mint a fenti adatokból is látszik, túl sok értelme nincs bemutatni, mert majdnem minden osztály lefedettsége 0%. Ezért csak azoknak az adatait írom le, melyeknek a lefedettsége magasabb mint 0%.

*org.jkiss.dbeaver.ui.data.dialogs.TextViewDialog*.java méret metrikái:

1. Lines of Code: 370
2. Lines: 454
3. Statements: 193
4. Functions: 18
5. Classes: 1
6. Files: 1
7. Comment Lines: 13
8. Comments (%): 3.4%

*org.jkiss.dbeaver.ui.data.dialogs.TextViewDialog*.java tényleges lefedettsége:

1. Coverage: 5.2%
2. Lines to Cover: 216
3. Uncovered Lines: 204
4. Lines Coverage: 5.6%

*org.jkiss.dbeaver.ui.data.dialogs.ValueViewDialog.java* méret metrikái:

1. Lines of Code: 299
2. Lines: 385
3. Statements: 102
4. Functions: 35
5. Classes: 2
6. Files: 1
7. Comment Lines: 13
8. Comments (%): 4.2%

*org.jkiss.dbeaver.ui.data.dialogs.ValueViewDialog.java* tényleges lefedettsége:

1. Coverage: 1.8%
2. Lines to Cover: 135
3. Uncovered Lines: 132
4. Lines Coverage: 2.2%

# Összegzés

Sajnos az egész csomag (illetve igaz ez a projektre) elég nehezen tesztelhető a sok függőség miatt, illetve a nem megfelelő kódminőség miatt. TextViewDialog 5% körüli lefedettsége és az új funkció (JSON beautifier) miatt létrehozott két statikus metódus teszteléséből fakadt, amelyek pont azért, mert statikusok, ezért tesztelhetőek voltak. Viszont a többi része az osztálynak, és a többi osztály már nem volt tesztelhető, mert nem lehetett példányosítani az adott osztályt. az általam bemutatott csomag, illetve valószínűleg az egész projekt egy nagyobb refaktorálásra szorulni, de ez eléggé nehézkes lenne a projekt mérete miatt. Így marad az alacsony tesztelhetőség. Ahol esetleg lehet javítani a lefedettségen, az a különböző Utility, illetve modell osztályok.