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.

A tesztek determinisztikus lefutását Mockito segítségével biztosítottam. Mockito egy open source tesztelő keretrendszer, amely lehetővé teszi a Java programozási nyelven íródott egységtesztekhez szükséges objektumok mock megfelelőinek létrehozását. Ezen kívül Mockito használatával előre meg tudjuk határozni az egyes külső, független metódusok viselkedését. Ezáltal az automatizált teszt minden lefutás során ugyanazt az eredményt produkálja.

Tesztelt osztályok

org.jkiss.dbeaver.model.impl.sql.SQLQueryTransformerSelectAllFrom.java: Az általam implementált „*SELECT * FROM <table_name>*” funkció egy fontos része az *SQLQueryTransformerSelectAllFrom* osztály, amely a megadott tábla lekérdezését végzi. Az osztály a betöltött szerepét tekintve egy rendkívül kritikus része a megvalósított funkciónak, ezért teszteltem külön.

org.jkiss.dbeaver.ui.editors.sql.util.SQLWordFinder.java: Az általam implementált „*SELECT * FROM <table_name>*” funkció egy másik, hasonlóan fontos része az *SQLWordFinder* osztály, amely a megadott tábla nevének azonosítását végzi. Ez azért egy kritikus része a funkciónak, hiszen a szövegszerkesztő kontextusából kell kinyerni egy létező adattábla nevet. Az osztályt a betöltött szerepe miatt teszteltem külön.

Létrehozott tesztcsomag bemutatása

org.jkiss.dbeaver.model.impl.sql.SQLQueryTransformerSelectAllFromTest.java: Az *SQLQueryTransformerSelectAllFrom* osztályon belüli *transformQuery()* metódus tesztelését végzi. Ezen metódus állítja össze a megadott tábla lekérdezéséhez szükséges SQL lekérdezést. Két esetet teszteltem. Az egyik az ideális futás, ahol ellenőriztem, hogy a megadott táblanév esetén helyesen jön-e létre az SQL lekérdezés. A másik esetben nem adtam meg a táblanevet és egy kivétel képződését vizsgáltam.

org.jkiss.dbeaver.ui.editors.sql.util.SQLWordFinderTest.java: Az *SQLWordFinder* osztályon belüli *findWord()*, *getWordStartOffset()* és *getWordEndOffset()* metódusok tesztelését végzi. A *getWordStartOffset()* metódus a megadott szöveg egy tetszőleges pontjából indulva az első lehetséges szó kezdetének pozícióját adja vissza. A tesztelés során külön vizsgáltam a metódus viselkedését szöveg megadása nélkül, a megadott szöveg vizsgálatát nem megfelelő pozícióból indítva, valamint szóközt tartalmazó és szóközt nem tartalmazó szövegek megadása esetén. A *getWordEndOffset()* a megadott szöveg egy tetszőleges pontjából indulva az első lehetséges szó végének pozícióját adja vissza. A tesztelés során külön vizsgáltam a metódus viselkedését szóközt tartalmazó, valamint szóközt nem tartalmazó szövegek megadása esetén. A *findWord()* metódus a megtalált szó régióját határozza meg, amely tartalmazza az adott szó elejének és végének pozícióit. Ez a metódus használja az előző kettőt, ezért külön hangsúlyt kellett fektetni a determinisztikus futásra. A tesztelés során külön vizsgáltam a metódus viselkedését szöveg megadása nélkül, valamint amikor a kezdeti és a végső pozíciók megegyeznek, és amikor nem.

Létrehozott tesztek pontos megadása

org.jkiss.dbeaver.model.impl.sql.SQLQueryTransformerSelectAllFromTest.java:

1. *testTransformQueryWhenIdentifierGiven()*:
    - Arrange: 16-19. sorok
    - Act: 20. sor
    - Assert: 21-20. sorok

2. *testTransformQueryWhenNoSpecifiedIdentifier()*:
    - Arrange: 27-30. sorok
    - Act: 31. sor
    - Assert: 32-33. sorok

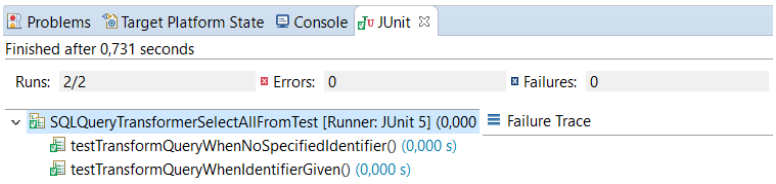
org.jkiss.dbeaver.model.impl.sql.SQLQueryTransformerSelectAllFromTest.java:

1. *findWordWhenDocumentIsEmpty()*:
    - Arrange: 15-16. sorok
    - Act: 17. sor
    - Assert: 18. sor
2. *findWordWhenStartAndEndIsEquals()*:
    - Arrange: 23-24. sorok
    - Act: 25. sor
    - Assert: 26-27. sorok
3. *findWordWhenStartAndOffsetIsNotEquals()*:
    - Arrange: 23-33. sorok
    - Act: 34. sor
    - Assert: 35-36. sorok
4. *testGetWordStartOffsetWhenTextIsNull()*:
    - Arrange: 41. sor
    - Act: 41. sor
    - Assert: 42-43. sorok
5. *testGetWordStartOffsetWhenStartIndexIsInvalid()*:
    - Arrange: 48. sor
    - Act: 48. sor
    - Assert: 49-50. sorok
6. *testGetWordStartOffsetWhenTextHasWhiteSpace()*:
    - Arrange: 55. sor
    - Act: 55. sor
    - Assert: 56-57. sorok

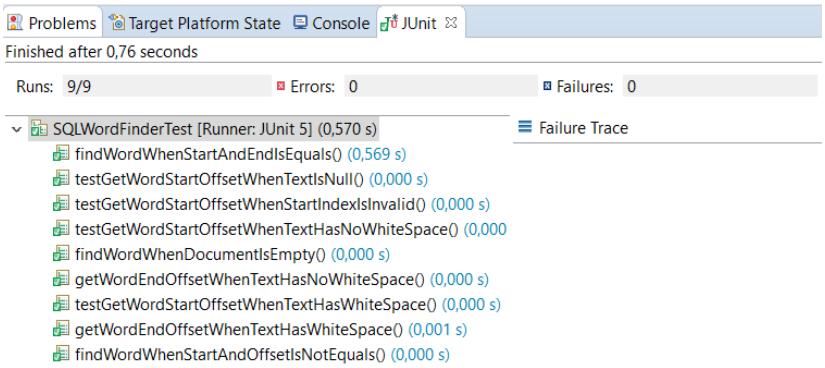
7. *testGetWordStartOffsetWhenTextHasNoWhiteSpace()*
    - Arrange: 62. sor
    - Act: 62. sor
    - Assert: 63-64. sorok
8. *getWordEndOffsetWhenTextHasWhiteSpace()*:
    - Arrange: 69. sor
    - Act: 69. sor
    - Assert: 70-71. sorok
9. *getWordEndOffsetWhenTextHasNoWhiteSpace()*:
    - Arrange: 76. sor
    - Act: 76. sor
    - Assert: 77-78. sorok

Végrehajtott tesztek sikerességének megállapítása

org.jkiss.dbeaver.model.impl.sql.SQLQueryTransformerSelectAllFromTest.java: Mind a két teszt sikeresen átment.



org.jkiss.dbeaver.model.impl.sql.SQLQueryTransformerSelectAllFromTest.java: Mind a kilenc teszt sikeresen átment.



Lefedettségi 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.editors.sql* csomag lefedettségét mértem. Mivel ezen csomaghoz egyetlen egységteszt sem tartozik, így nem nehéz megállapítani, hogy a lefedettsége 0%.

org.jkiss.dbeaver.ui.editors.sql csomag méret metrikái:

1. Lines of Code: 18,060
2. Lines: 25,362
3. Statements: 7,842
4. Functions: 1,233
5. Classes: 215
6. Files: 151
7. Comment Lines: 1,335
8. Comments (%): 6.9%

org.jkiss.dbeaver.ui.editors.sql csomag tényleges lefedettsége:

1. Coverage: 0.0%
2. Lines to Cover: 9,229
3. Uncovered Lines: 9,229
4. Lines Coverage: 0.0%

Ez az érték azért lehet probléma, mert az alkalmazás egy nagyon fontos funkciójának a magjáról beszélünk, ugyanis ez a csomag a szíve az SQL szerkesztő felületnek. A lefedettségi értékekből ítélve viszont megkérdőjelezhető ezen funkció tényleges működése.

Erősen és kevésbe lefedett kódelemek bemutatása

Az *SQLEditorBase.java* és az *SQLEditor.java* két kulcsfontosságú osztály. Ezen osztályok lefedése lényeges kódminőség javuláshoz vezethetne. Ez a két osztály része volt az általam fejlesztett „*SELECT * FROM <table_name>*” funkciónak. Sajnos mivel az egész csomag lefedettsége is aggasztó volt, így ezen két osztályé is.

org.jkiss.dbeaver.ui.editors.sql.SQLEditorBase.java méret metrikái:

1. Lines of Code: 674
2. Lines: 887
3. Statements: 316
4. Functions: 61
5. Classes: 3
6. Files: 1
7. Comment Lines: 75
8. Comments (%): 10.0%

org.jkiss.dbeaver.ui.editors.sql.SQLEditorBase.java tényleges lefedettsége:

1. Coverage: 0.0%
2. Lines to Cover: 381
3. Uncovered Lines: 381
4. Lines Coverage: 0.0%

org.jkiss.dbeaver.ui.editors.sql.SQLEditor.java méret metrikái:

9. Lines of Code: 3,013
10. Lines: 3,543
11. Statements: 1,631
12. Functions: 169
13. Classes: 15
14. Files: 1
15. Comment Lines: 146
16. Comments (%): 4.6%

org.jkiss.dbeaver.ui.editors.sql.SQLEditor.java tényleges lefedettsége:

5. Coverage: 0.0%
6. Lines to Cover: 1,817
7. Uncovered Lines: 1,817
8. Lines Coverage: 0.0%

Könnyen és nehezen tesztelhető funkciók és változások

A korábban említett két kulcsfontosságú osztály tesztelése sajnos nem valósulhatott meg, ugyanis a példányosításuk, mock-olásuk nem volt lehetséges. Ez a kódminőségnek tudható be, ugyanis ezen osztályokban számos kódrész szerepel, amelyeket nem lehet felülírni. Ezenkívül rengeteg függőséggel rendelkeznek, melyek mock-olása jóformán lehetetlen. Ezek és az ezekhez hasonló osztályok esetében először a kódminőség javítására kellene fókuszálni, hiszen anélkül nehéz lesz tesztelni és tesztelés által lefedettséget növelni. A lefedettségi érték tökéletesen tükrözi ezen gondokat.

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.

A tesztek determinisztikus lefutását Mockito segítségével biztosítottam. Mockito egy open source tesztelő keretrendszer, amely lehetővé teszi a Java programozási nyelven íródott egységtesztekhez szükséges objektumok mock megfelelőinek létrehozását. Ezen kívül Mockito használatával előre meg tudjuk határozni az egyes külső, független metódusok viselkedését. Ezáltal az automatizált teszt minden lefutás során ugyanazt az eredményt produkálja.

Tesztelt osztályok

org.jkiss.dbeaver.ui.UIUtils.java: Az UIUtils.java osztály tesztelését választottam. Az osztály a betöltött szerepét tekintve egy rendkívül kritikus metódusokat valósít meg, amik a felhasználói felület felépítéséhez szükséges függvények, ezért teszteltem külön.

Létrehozott tesztcsomag bemutatása

org.jkiss.dbeaver.ui.UIUtilsTest.java: A *UIUtilson* osztályon belüli *getColorByRGB(), getSharedColor(), greyLevel(), getContrastColor(),* metódusok tesztelését végzi. A *getColorByRGB()* metódus a paraméterben megadott string alapján adja vissza az adott eszköz színét. A tesztelés során külön vizsgáltam a metódus viselkedését string megadása nélkül, helyes string megadásával, illetve hibás paraméter átadása esetén is. A *getSharedColor()* metódus a paraméterben megadott string alapján visszaadja a megosztott színeket. A tesztelés során külön vizsgáltam üres string átadásával, illetve megfelelő paraméterek átadása esetén is. A *greyLevel()* metódus visszaadja azt a szürke értéket, amelyen a megadott szín kirajzolódna a szürke-skálán. A tesztelés során helye paraméter átadásával és null paraméter átadásával vizsgáltam a viselkedést. A getContrastColor() metódus kiszámolja a kontrasztot a Luma fényerő alapján. A metódust tesztelése során külön vizsgáltam 0.5-ös luma határértéknél kisebb, illetve nagyobb számra.

Létrehozott tesztek pontos megadása

org.jkiss.dbeaver.ui.UIUtilsTest.java:

3. *testGetColorRGBWithValid()*:
    - Arrange: 18-22. sorok
    - Act: 23. sor
    - Assert: 24. sor

org.jkiss.dbeaver.ui.UIUtilsTest.java:

1. *testGetColorRGBWithNull()*:
    - Arrange: 29. sor
    - Act: 30. sor
    - Assert: 30. sor

org.jkiss.dbeaver.ui.UIUtilsTest.java:

1. *testGetColorRGBWithInvalid()*:
    - Arrange: 35-36. sorok
    - Act: 37. sor
    - Assert: 38. sor

org.jkiss.dbeaver.ui.UIUtilsTest.java:

1. *testSharedColorWithNull()*:
    - Arrange: 43 sor
    - Act: 44. sor
    - Assert: 44. sor

org.jkiss.dbeaver.ui.UIUtilsTest.java:

1. *testSharedColorWithValid ()*:
    - Arrange: 49-53 sor
    - Act: 54. sor
    - Assert: 55. sor

org.jkiss.dbeaver.ui.UIUtilsTest.java:

1. *testgreyLevelEqual ()*:
    - Arrange: 60. sor
    - Act: 61. sor
    - Assert: 61. sor

org.jkiss.dbeaver.ui.UIUtilsTest.java:

1. *testgreyLevelWithValid ()*:
    - Arrange: 66-67. sorok
    - Act: 68. sor
    - Assert: 68. sor

org.jkiss.dbeaver.ui.UIUtilsTest.java:

1. *testgetContrastColorWithNull ()*:
    - Arrange: 73-74. sorok
    - Act: 75. sor
    - Assert: 75. sor

org.jkiss.dbeaver.ui.UIUtilsTest.java:

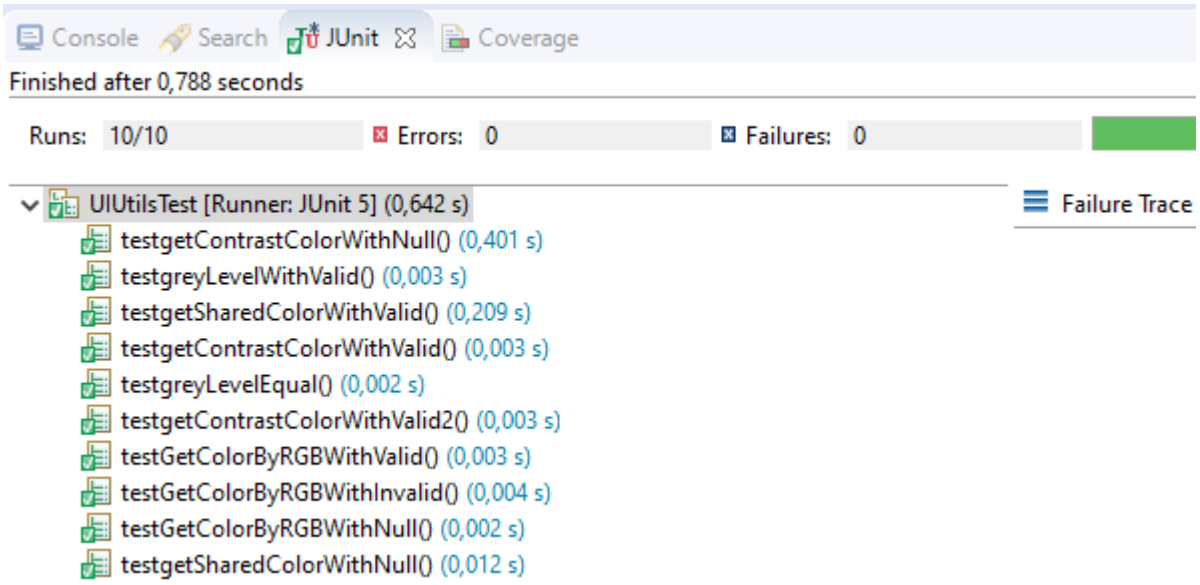
1. *testgetContrastColorWithValid ()*:
    - Arrange: 80-84. sorok
    - Act 85. sor
    - Assert: 85. sor

org.jkiss.dbeaver.ui.UIUtilsTest.java:

1. *testgetContrastColorWithValid2 ()*:
    - Arrange: 90-94. sorok
    - Act: 95. sor
    - Assert: 95. sor

Végrehajtott tesztek sikerességének megállapítása

org.jkiss.dbeaver.ui.UIUtilsTest.java: Mind a tíz teszt sikeresen átment.



Lefedettségi jelentés

Lefedettség mérés során felhasznált eszközök

A kód lefedettséget SonarQube segítségével mértem. A SonarQube egy open source szoftver, amely a forráskód minőségének ellenőrzésére és követésére szolgál. A SonarQube 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

Az én választásom az *org.jkiss.dbeaver.core* csomagra esett. A csomag lefedettsége tesztek hiányában 0%

org.jkiss.dbeaver.core csomag méret metrikái:

9. Lines of Code: 16,246
10. Lines: 21,813
11. Statements: 6,883
12. Functions: 957
13. Classes: 164
14. Files: 128
15. Comment Lines: 876
16. Comments (%): 5.1%

org.jkiss.dbeaver.core csomag tényleges lefedettsége:

5. Coverage: 0.0%
6. Lines to Cover: 8,085
7. Uncovered Lines: 8,085
8. Lines Coverage: 0.0

A legtöbb alapvető felhasználó felületi osztályt (UI) ez a csomag tartalmazza. A lefedettség értéke 0.0%, ami már magában is egy elég beszédes szám és nagy problémát jelenthet. A lefedettségi értékekből ítélve megkérdőjelezhető a különböző funkciók megfelelő működése.

Erősen és kevésbe lefedett kódelemek bemutatása

org.jkiss.dbeaver.core.ui.dialogs.driver.EditDialog.java méret metrikái:

1. Lines of Code: 750
2. Lines: 900
3. Statements: 433
4. Functions: 38
5. Classes: 3
6. Files: 1
7. Comment Lines: 43
8. Comments (%): 5.4%

org.jkiss.dbeaver.core.ui.dialogs.driver.EditDialog.java tényleges lefedettsége:

1. Coverage: 0.0%
2. Lines to Cover: 493
3. Uncovered Lines: 493
4. Lines Coverage: 0.0%

org.jkiss.dbeaver.core.ui.dialogs.driver.DriverManagerDialog.java méret metrikái:

9. Lines of Code: 343
10. Lines: 475
11. Statements: 180
12. Functions: 17
13. Classes: 1
14. Files: 1
15. Comment Lines: 56
16. Comments (%): 14%

org.jkiss.dbeaver.core.ui.dialogs.driver.DriverManagerDialog.java tényleges lefedettsége:

5. Coverage: 0.0%
6. Lines to Cover: 203
7. Uncovered Lines: 203
8. Lines Coverage: 0.0%

Könnyen és nehezen tesztelhető funkciók és változások

A projekten belüli kulcsfontosságú osztályok tesztelése sajnos nem valósítható meg, ugyanis, ezen osztályok példányosítása, mockolása nem volt lehetséges. Ez főként a kódminőségnek köszönhető. Ezen osztályokban rengeteg olyan kódrész szerepel, amelyeket nem lehet felülírni. Ezenkívül rengeteg függőséggel rendelkeznek, melyek mock-olása jóformán lehetetlen. Ezek és az ezekhez hasonló osztályok esetében először a kódminőség javítására kellene fókuszálni, hiszen anélkül nehéz lesz tesztelni és tesztelés által lefedettséget növelni.

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

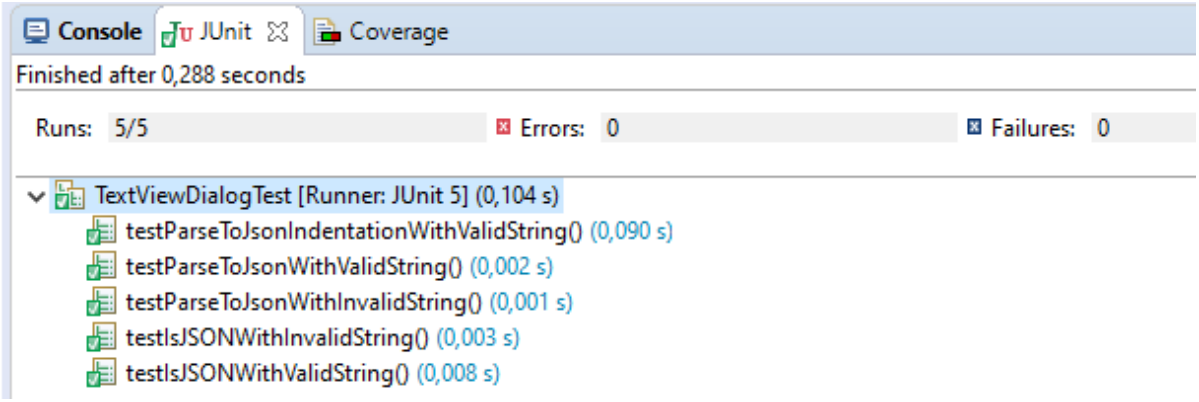
org.jkiss.utils.xml.XMLUtilsTest.java

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

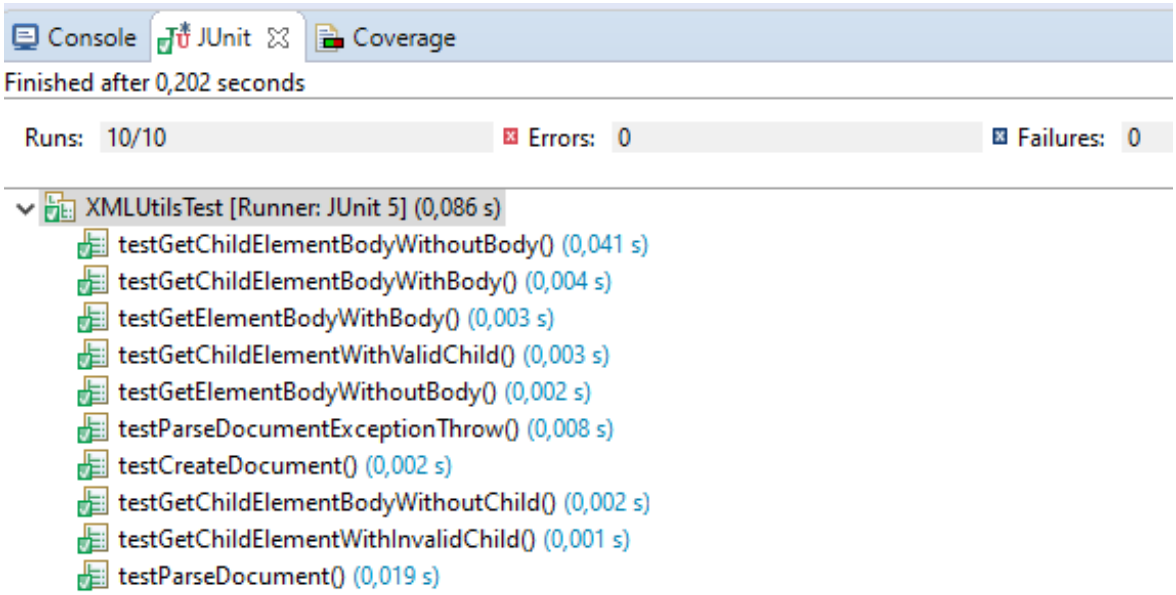
4. testGetChildElementWithValidChild():
    - Arrange: 37, 76-80. sorok
    - Act: 81. sor
    - Assert: 82-83. sorok
5. testGetChildElementWithInvalidChild():
    - Arrange: 35, 92-96. sorok
    - Act: 97. sor
    - Assert: 98. sor
6. testGetElementBodyWithBody():
    - Arrange: 38, 108-112. sorok
    - 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:

9. Lines of Code: 299
10. Lines: 385
11. Statements: 102
12. Functions: 35
13. Classes: 2
14. Files: 1
15. Comment Lines: 13
16. Comments (%): 4.2%

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

5. Coverage: 1.8%
6. Lines to Cover: 135
7. Uncovered Lines: 132
8. 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.

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.

A tesztek determinisztikus lefutását Mockito segítségével biztosítottam. Mockito egy open source tesztelő keretrendszer, amely lehetővé teszi a Java programozási nyelven íródott egységtesztekhez szükséges objektumok mock megfelelőinek létrehozását. Ezen kívül Mockito használatával előre meg tudjuk határozni az egyes külső, független metódusok viselkedését. Ezáltal az automatizált teszt minden lefutás során ugyanazt az eredményt produkálja.

Tesztelt osztályok

org.jkiss.dbeaver.utils.GeneralUtils.java: A GeneralUtils.java osztály tesztelését választottam. Ez az osztály definiál nagyon sok általános metódust és éppen ezért fontosak mert sokat használtak a progam működése során.

Létrehozott tesztcsomag bemutatása

org.jkiss.dbeaver.utils.GeneralUtilsTest.java: A GeneralUtils osztalyban talalható convetToString(), makeDisplayString(), convertToBytes(), varriablePattern(), GetRootCause(), isVarriablePattern(), metódusokat teszteli. A convertToString() egy byte tömböt alakít szöveggé, úgy hogy kap egy offsetet és egy hosszat is hozzá hogy hol kezdje és mekkora részt alakítson át belőle. A makeDisplayString() felel azért hogy bármely obijektumot ki lehessen íratni. A convertToBytes() egy Stringből készíti el a hozzá tartozó byte tömböt. A varriablePattern() egy nevet kap és ahhoz létrehozza a pattern stringet.. A getRootCause() visszadja egy dobott hiba eredetét. Az Is VarriablePattern() elnevezési konvenciót ellenőriz.

Létrehozott tesztek pontos megadása

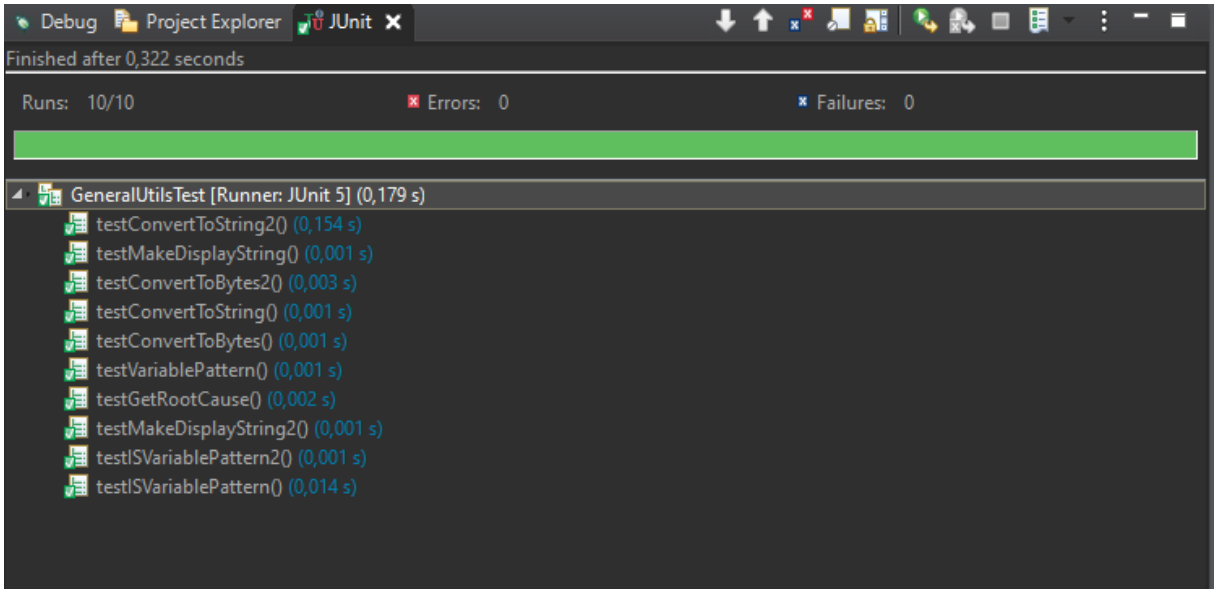
org.jkiss.dbeaver.utils.GeneralUtilsTest.java:

1. *testCovertToString2()*:
    - Arrange: 18-22. sorok
    - Act: 23. sor
    - Assert: 23. sor
2. *testMakeDisplayString()*:
    - Arrange: 59. sor
    - Act: 60. sor
    - Assert: 60. sor
3. *testConvertToBytes2()*:
    - Arrange: 34-35. sorok
    - Act: 36. sor
    - Assert: 37. sor
4. *testConvertToString()*:
    - Arrange: 11-14. sor
    - Act: 15. sor
    - Assert: 15. sor

5. *testConvertToBytes()*:
    - Arrange: 27-28. sor
    - Act: 29. sor
    - Assert: 30. sor
6. *testVarriablePattern()*:
    - Arrange: 41. sor
    - Act: 42. sor
    - Assert: 43. sor
7. *testGetRootCause()*:
    - Arrange: 70. sorok
    - Act: 71. sor
    - Assert: 72. sor
8. *testMakeDisplayString2()*:
    - Arrange: 64. sorok
    - Act: 65. sor
    - Assert: 66. sor
9. *testIsVarriablePattern2 ()*:
    - Arrange: 53. sorok
    - Act 54. sor
    - Assert: 55. sor
10. *testIsVarriablePattern ()*:
    - Arrange: 47. sorok
    - Act: 48. sor
    - Assert: 49. sor

Végrehajtott tesztek sikerességének megállapítása

org.jkiss.dbeaver.utils.GeneralUtilsTest.java: Mind a tíz teszten sikeresen átment.



Lefedettségi jelentés

Lefedettség mérés során felhasznált eszközök

A kód lefedettséget SonarQube segítségével mértem. A SonarQube egy open source szoftver, amely a forráskód minőségének ellenőrzésére és követésére szolgál. A SonarQube 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

Az én választásom az *org.jkiss.dbeaver.model* csomagra esett. A csomag lefedettsége tesztek hiányában 0.0%

org.jkiss.dbeaver.model csomag méret metrikái:

1. Lines of Code: 21,411
2. Lines: 79,317
3. Statements: 17,311
4. Functions: 6,407
5. Classes: 784
6. Files: 710
7. Comment Lines: 3641
8. Comments (%): 6.6%

org.jkiss.dbeaver.core csomag tényleges lefedettsége:

1. Coverage: 0.0%
2. Lines to Cover: 21,411
3. Uncovered Lines: 21,411
4. Lines Coverage: 0.0%

A model felel a legtöbb alapműveletér és ez a lefedettség nagyon rosszat jelent a megbízhatóságáról.

Erősen és kevésbe lefedett kódelemek bemutatása

org.jkiss.dbeaver.model.DBUtils.java méret metrikái:

17. Lines of Code: 1,783
18. Lines: 2,086
19. Statements: 959
20. Functions: 125
21. Classes:
22. Files: 1
23. Comment Lines: 96
24. Comments (%): 5.1%

org.jkiss.dbeaver.model.DBUtils.java tényleges lefedettsége:

9. Coverage: 0.0%
10. Lines to Cover: 1,022
11. Uncovered Lines: 1,022
12. Lines Coverage: 0.0%

org.jkiss.dbeaver.model.virtual.DBVEntity.java méret metrikái:

25. Lines of Code: 603
26. Lines: 704
27. Statements: 311
28. Functions: 57
29. Classes: 1
30. Files: 1
31. Comment Lines: 10
32. Comments (%): 1.6%

org.jkiss.dbeaver.core.ui.dialogs.driver.DriverManagerDialog.java tényleges lefedettsége:

1. Coverage: 0.0%
2. Lines to Cover: 370
3. Uncovered Lines: 370
4. Lines Coverage: 0.0%

Összegzés

Az egész projekt lefedettsége szörnyű elsőnek ezen kellene javítani. A projektben szinte semmi sincs kommentálva vagy dokumentálva így nehéz dolgozni vele nagy eséllyel emiatt van ilyen állapotban. A dokumentációk alapján lehetne javítani a projekt minőségét és azzal a tesztelést is elérhetőbbé tenni. Így ezen állapotában túl nehezen tesztelehető a projekt.