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 határétéknél kisebb, illetve nagyobb számra.

# Létrehozott tesztek pontos megadása

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

1. *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

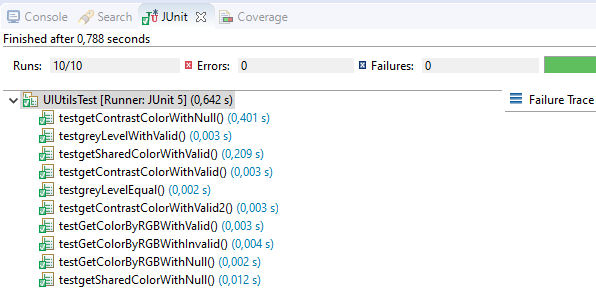
*org.jkiss.dbeaver.ui.UIUtilsTest.java*:

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:*

1. Lines of Code: 16,246
2. Lines: 21,813
3. Statements: 6,883
4. Functions: 957
5. Classes: 164
6. Files: 128
7. Comment Lines: 876
8. Comments (%): 5.1%

*org.jkiss.dbeaver.core* csomag tényleges lefedettsége:

1. Coverage: 0.0%
2. Lines to Cover: 8,085
3. Uncovered Lines: 8,085
4. 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:

1. Lines of Code: 343
2. Lines: 475
3. Statements: 180
4. Functions: 17
5. Classes: 1
6. Files: 1
7. Comment Lines: 56
8. Comments (%): 14%

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

1. Coverage: 0.0%
2. Lines to Cover: 203
3. Uncovered Lines: 203
4. 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.