Eclipse - JUnit 5-testien lausekattavuus

Tehtävässä tutustutaan Eclipsen JUnit 5-testausympäristöön ja selvitetään testien kattavuus.

1) Luo Maven-projekti ja lisää pom.xml:ään testauksessa tarvittavat määrittelyt

2) Pura tiedosto

   http://users.metropolia.fi/~hakka/OTP1/Laskin5.zip

ja raahaa pakkauksesta purkamasi koodit (src, test) drag&dropilla projektiisi.

Jos skandimerkit eivät näy oikein, vaihda merkistöä Eclipse-ympäristön Workspace-tasolla.
- valitse Window | Preferences | General | Workspace
- aseta Text file encoding UTF-8

TEHTÄVÄN OSA 1: Kattavuus

Aja sovellus (käynnistä Main.java), ja totea tulostuksista, että toteutuksessa on virheitä.
- klikkaa hiiren oikealla Project Explorerissa tiedostonimeä Main.java
- valitse Run As | Java Application tai näppäile Alt+Shift+X ja A.

Aja sitten JUnit-testit
- klikkaa hiiren oikealla Project Explorerissa projektin nimeä
- valitse Run As | JUnit Test tai näppäile Alt+Shift+X ja T.
- Yksittäisen JUnit-testiluokan voi ajaa vastaavasti klikkaamalla sen nimeä.

Kuten huomaat, osa testeistä menee läpi, osa ei.

Ennenkuin lähdet korjaamaan testejä / koodeja, niin selvitä aluksi millainen lausekattavuus mukana olevilla JUnit-testeillä saavutetaan. Aja siis kattavuusanalyysi
- klikkaa Package Explorerissa projektin nimeä hiiren oikealla
- valitse Coverage As -kohdasta joko ”1 Java Application” tai ”2 JUnit Test”. Seurata voi joko sovelluksen suoritusaikaista kattavuutta tai JUnit-testien kattavuutta.

Analyysin voi käynnistää myös Eclipsen yläreunan suorita-kuvakkeista 

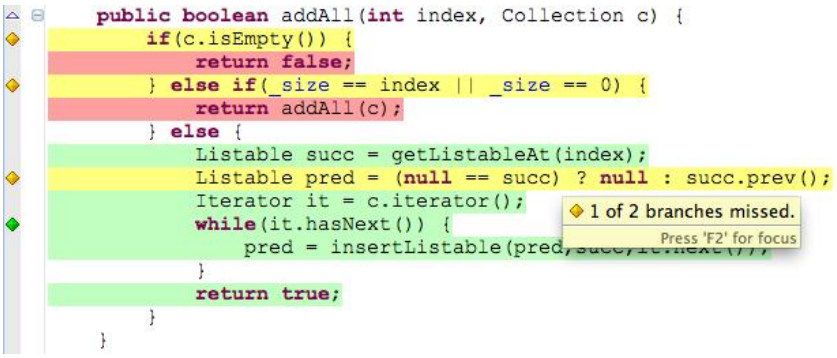
Yhteenveto ilmestyy ruudun alareunan Coverage-välilehdelle. Tarvittaessa näkymän voi avata valitsemalla Window | Show View | Other... ja edelleen Java | Coverage. Näkymästä käy ilmi kuinka suuri osa kunkin tiedoston lauseista tuli suoritettua.

Console | Coverage

junitHarj (29.1.2020 11:51:11)

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
junitHar	50,0 %	161	161	322
src/main/java	29,7 %	43	102	145
(default package)	0,0 %	0	101	101
laskin	97,7 %	43	1	44
Laskin.java	97,7 %	43	1	44
Laskin	97,7 %	43	1	44
kerro(int)	0,0 %	0	1	1
annaTulos()	100,0 %	3	0	3
jaa(int)	100,0 %	7	0	7
lisaa(int)	100,0 %	7	0	7
nelio(int)	100,0 %	6	0	6
neliojuuri(int)	100,0 %	1	0	1
nollaa()	100,0 %	4	0	4
vahenna(int)	100,0 %	7	0	7
virtaOFF()	100,0 %	1	0	1
virtaON()	100,0 %	4	0	4
src/test/java	66,7 %	118	59	177
laskin	66,7 %	118	59	177
LaskinTest.java	63,8 %	44	25	69
ExtraTest.java	75,3 %	55	18	73
NelioTest.java	0,0 %	0	16	16
AbstractParent.java	100,0 %	19	0	19

Kun testit on suoritettu, analyysin tulos on tutkittavissa lausekohtaisesti testattaviin luokkiin merkityistä värillisistä korostuksista (kuva ei tästä projektista):



Vihreiksi merkityt koodikokonaisuudet on suoritettu kokonaan, keltaisella merkityt osat vain osittain (esim. pelkkä valinnan ehto-osa tai vain osa ehdoista), ja punaisissa osissa ei ole käyty lainkaan.

Yhteenvedon mukaan kattavuus näyttäisi olevan hyvä, vaikka JUnit-testeissä onkin paljon puutteita. Kattavuus kertoo kuitenkin vain, paljonko testattavan koodin lauseista on suoritettu, mutta se ei sinänsä kerro mitään muuta testaamisen laadusta.

Kattavuusraportin voi tallettaa html-sivustona seuraavasti

- klikkaa projektin nimeä hiiren oikealla ja valitse Export
- valitse Run/Debug ja Coverage Session
- anna hakemisto, jonne raporttisivusto tallennetaan (esim.target/coverage-report)
- pääset tutkimaan raporttia klikkaamalla tiedostoa index.html.

Eclipsen EclEmma-lisäosasta löytyy infoa osoitteella http://www.eclemma.org/.
Tutustu myös näihin sivustoihin http://en.wikipedia.org/wiki/Code_coverage ja http://en.wikipedia.org/wiki/Java_Code_Coverage_Tools

TEHTÄVÄN OSA 2: JUnit 5

Koodissa ja testeissä on tarkoituksellisia virheitä ja puutteita. Korjaa koodi siten, että testit eivät enää löydä yhtään virhettä. Varmista myös, että testaus on kattavaa - tee tarvittaessa lisää testimetodeja.

1) Aja testit ja korjaa testeissä löytyvät, esimerkkiin tarkoituksella sijoitetut virheet.

2) Täydennä puuttuvat metodien osat ja metodit (mm. `kerro()` ja `neliojuuri()`). Kutsu metodin `neliojuuri()` toteutuksessa kirjastometodia `Math.sqrt()`, ja muunna (cast) tulos int-tyyppiseksi. Korjaa laskinta siten, että nollalla jakaminen ei ole mahdollista ja että negatiivisesta luvusta ei voi ottaa neliöjuurta. Kummassakin tapauksessa metodin tulee heittää IllegalArgumentException-poikkeus, jonka kutsuja voi käsitellä haluamallaan tavalla. Nollallajaossa poikkeuksen ilmoituksen tulee olla ”Nollalla ei voi jakaa” ja neliöjuuren poikkeuksen ilmoituksen tulee olla ”Negatiivisella luvulla ei ole neliöjuurta”.

3) Luokassa `ExtraTest` on keskeneräinen testausmetodi metodille `neliojuuri()`. Täydennä ja virittele lisää testejä (mm. negatiivinen syöte).

Testimetodeihin (`AbstractParent` ja `ExtraTest`) on laitettu ylimääräisiä tulostuksia, jotka osoittavat milloin annotaatiolla `@BeforeAll`, `@BeforeEach`, `@AfterEach` ja `@AfterAll` merkityt metodit suoritetaan. Katso tulostukset ja lue koodi ajatuksella. Mistä yksittäinen tulostusrivi on tullut?

Katso Console-välilehdeltä mitä nämä testit tulostavat stdout-virtaan. Jos Console-ikkuna ei näy ruudun alareunassa, niin valitse Window | Show View | Console (Alt+Shift+Q, C).

ParameterizedTest

Luokka `ExtraTest` testaa metodin `nelio()` toimintaa usealla metodilla (`testNelio2()`, `testNelio4()`, `testNelio5()`). Tällaisen copy-paste -tekniikan sijasta testitapaukset on parempi välittää testiajurille taulukoituina parametreina.

Tästä on esimerkki luokassa `NelioTest`. Luokassa on vain yksi testimetodi. Se poikkeaa aiemmista siten, että tällä metodilla on kaksi parametria. JUnit 5 suorittaa annotaatiolla @ParameterizedTest merkityn testimetodin useita kertoja, ja kullakin kerralla kutsussa on eri todelliset parametrit. Yksittäisen testin saamat parametrit (tässä syöte ja odotettu tulos) on lueteltu @CsvSource-annotaatiolle (comma separated values) annetun taulukon yksittäisessä alkiossa. @ParameterizedTest suorituttaa testimetodin kerran jokaisella taulukon parametriparilla.

Parametroiduista testeistä ja muista parametrin antotavoista (argument sources) löytyy lisätietoa esimerkiksi osoitteesta blog.codefx.org/libraries/junit-5-parameterized-tests/

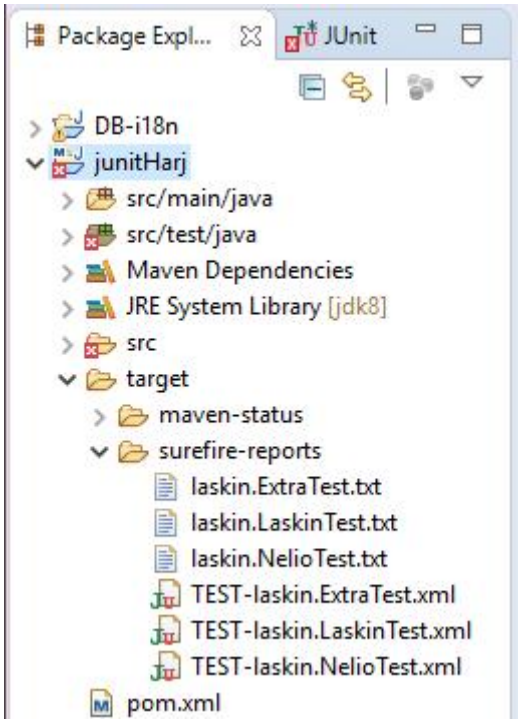
4) Muuta laskinta siten, että laskin osaa käsitellä double-tyyppisiä reaalilukuja. Kun assert-rutiineissa verrataan reaalilukuja, on rutiinien kolmanneksi parametriksi annettava ns. delta-arvo, joka ilmaisee kuinka paljon arvot saavat poiketa toisistaan. Ks. JUnit API Assertions.

   Muista: Kun muutat koodia, on myös testien oikeellisuus ja kattavuus aina varmistettava. Regressiotestaa koodisi muutosten jälkeen (aja testisarjasi) niin, että lopulta kaikki testit hyväksytään.

5) Tarkasta vielä testien lausekattavuus. Ja mieti vielä kerran tulitko testanneeksi oikeat asiat oikealla tavalla.

6) Suorituta testit vielä myös Mavenilla. Nyt testit suoritetaan IDE:n ulkopuolella (ns. eräajona ilman käyttäjälle kohdistuvaa I/O:ta). Tällainen tarve on mm. jatkuvassa koonnissa Jenkinsillä. Klikkaa projektin nimeä hiiren oikealla ja valitse `Run As | Maven test`.

   Nyt testien tulokset menevät projektin hakemiston target/surefire-reports.



Voit katsoa tuloksia klikkaamalla noita tiedostonimiä. Ota vastaava kuvakaappaus (esimerkiksi Windowsin Snipping Tool -työkalulla) ja liitä se tehtäväpalautukseen.