Metropolia

Oliosovellukset ja tietokannat

TEHTÄVÄSARJA 7

Kevät 2022

JUnit 5 - LASKIN

Kopioi tehtäväsarjan (**OlsoTehtSarja7**) koodit kurssin tietovarastosta omaan repoosi.

Sekä koodissa että testeissä on tarkoituksellisia virheitä ja puutteita. Tehtävänä on tutustua hiukan JUnit 5-ympäristöön ja korjata koodia sekä testausta siten, että enää löydy yhtään virhettä. Varmista myös, että testaus on kattavaa - tee tarvittaessa lisää testimetodeja.

Voit ajaa Eclipsessä yksittäisen JUnit-testiluokan napsauttamalla Project Explorerissa sen nimeä hiiren oikealla ja valitsemalla Run As | JUnit Test tai valitsemalla tiedoston ja näppäilemällä Alt+Shift+X ja T. Kaikki testit voi ajaa napsauttamalla vastaavasti projektin nimeä.

1) Aja sovellus (käynnistä Main.java) ja totea tulostuksista, että toteutuksessa on virheitä. Aja sitten testit ja korjaa testeissä löytyvät, esimerkkiin tarkoituksella sijoitetut virheet.

2) Täydennä puuttuvat metodien osat ja metodit (mm. `kerro()` ja `neliojuuri()`). Kutsu metodin `neliojuuri()` toteutuksessa tässä vaiheessa kirjastometodia `Math.sqrt()`, ja muunna (cast) tulos int-tyyppiseksi. Korjaa laskinta siten, että nollalla jakaminen ei ole mahdollista, eikä negatiivisesta luvusta voi ottaa neliöjuurta. Näissä tilanteissa 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”.

   Aja taas testit.

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 testeissä tulostetaan stdout-virtaan. Jos Console ei näy ruudun alareunassa, niin valitse Window | Show View | Console (Alt+Shift+Q, C).

Parameterized-ajuri

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

Tästä on esimerkki luokassa `NelioTest`. Siellä on vain yksi testimetodi. Se poikkeaa aiemmasta 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/

Tehtävän 4 voi kirjata tehdyksi, kun on tutustunut koodiin, suorittanut sen ja tutustunut tuohon blogikirjoituksen.

5) 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 5 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.

   Mieti vielä kerran tulitko testanneeksi oikeat asiat oikealla tavalla. Korjaa puutteet.

o-o-o