Harjoitus 6 (viikko 49)

Nämä ovat kurssin viimeiset harjoitukset. Muista, että hyväksytyistä ratkaisuista kerättyjen pisteiden summan tulee olla vähintään 60 % (26 pistettä) tehtävien kokonaismäärästä (42 kpl).

Voit tehdä lisätehtäviä (korkeintaan 8 kpl), jos aiot suorittaa kurssin, mutta pisteesi ovat alle 60 % -rajan tämän harjoituksen jälkeen. Kurssin vastuuopettaja ottaa yhteyttä opiskelijoihin, jotka voivat saavuttaa 60 %:n rajan lisätehtäviä tekemällä. Lisätehtävät ovat lisäpalvelu, joka paikka kiireellisyysasteikossa on vastuuopettajan pakollisten työtehtävien jälkeen. Arvosanan saa nopeammin, kun tekee vaaditun määrän tehtäviä normaalien harjoitusten puitteissa.

- Osa tämän harjoituksen tehtävistä liittyy joko suoraan tai epäsuorasti kurssin harjoitustyöhön. Harjoitustyö edistyy sitä paremmin, mitä enemmän tehtäviä ratkaiset.
- Mikäli tehtävissä on jotain epäselvää, laita sähköpostia vastuuopettajalle (jorma.laurikkala@tuni.fi).
- Muista nimetä muuttujat hyvin sekä kommentoida ja sisentää koodisi. Kirjoita metodin otsikkoon liittyvä yleisluonteinen kommentti metodin tarkoituksesta sekä metodin mahdollisesti saamista ja palauttamista tiedoista. Tee ohjelman alkuun kommentti, jossa on harjoituksen ja tehtävän numero sekä nimesi ja sähköpostiosoitteesi.
- Lue syötteet `Scanner`-luokalla. Käytä vain yhtä `Scanner`-oliota. Esittele lukija vakiomuotoisena attribuuttina, jos syötteitä on luettava kahdessa tai useammassa metodissa (katso luentorungon luku 2.2).
- Tehtäviin on tarjolla apua Zoom-ohjauksissa perjantaina 18.12. klo 10–12 ja maanantaina 21.12. klo 12–14. Näissä ohjauksissa voi kysyä myös harjoitustyöstä. Linkit Zoom-kokouksiin julkaistaan WETOssa.
- Palauta vastauksesi WETO-järjestelmään (https://wetodev.sis.uta.fi/) viimeistään ~~ensi viikon tiistaina klo 22.12. klo 23.55~~ torstaina 28.1.2020 klo 23.55.
- WETO tarkistaa kaikki tämän harjoituskerran tehtävät automaattisesti. Automaattisesta tarkistuksesta kerrotaan lisää kurssisivujen *Harjoitukset* | *Ratkaisujen tarkistus* | *Automaattinen* -kohdassa. Ota yhteyttä harjoitusryhmän vetäjään, jos et keksi järjellisessä ajassa miksi WETO hylkää palautuksesi.
- Kaikki opettajien tarkistamat tehtävät arvioidaan hyvän ohjelmointitavan osalta. Näin esimerkiksi huono sisennys tai puutteellinen kommentointi voi tuottaa nollan, vaikka ohjelma läpäisee WETOn testit. Muista erityisesti metodin otsikkoon liittyvä yleisluonteinen kommentti. Opettaja antaa paluuarvollisen oman metodin sisältävälle ratkaisulle pisteen vain, jos metodin paluuarvo sijoitetaan muuttujaan. Oman metodin kutsuja on oltava kussakin tehtävässä pyydetty määrä, jotta pisteen voi saada.

1. Kirjoita Javalla *CharacterArray2DCopier*-niminen ohjelma, jolla on *kopioi2dTaulukko*-niminen metodi kaksiulotteisen merkkitaulukon kopiointiin. Kopioitava taulukko välitetään metodille parametrina (`char[][]`). Metodi luo uuden samankokoisen taulukon ja kopioi parametritaulukon merkit uuteen taulukkoon, jos taulukko on vähintään 1×1 -kokoinen. Kukin merkki sijoitetaan uudessa taulukossa samaan paikkaan mistä se kopiointiin parametritaulukosta. Metodi palauttaa viitteen (`char[][]`) uuteen taulukkoon. Paluuarvo on `null`, jos parametrina saatu taulukko on `null`-arvoinen tai liian pieni. Metodi ei lue syötteitä eikä tulosta mitään.

   Kopioi tehtävän 5.2 tulostusmetodi *tulosta2d* ohjelmaan. Kutsu metodiasi vähintään viidesti pääohjelmassa. Käytä kutsuissa erilaisia parametriarvoja. Anna yhdessä kutsussa taulukkoparametrin arvoksi `null`. Sijoita kutsujen palauttamat arvot muuttujiin ja tulosta kopioidut taulukot antamalla muuttujat *tulosta2d*-metodin kutsujen parametriarvoiksi.

   Älä käytä *Arrays*-luokkaa.

2. Tee Javalla kokonaisluvuilla laskeva nelilaskin. Ohjelmaa käytetään antamalla sille syötteitä komentoikkunan tapaan. *Add*-, *diff*-, *multi*- ja *div*-komennot saavat parametreinaan aritmeettisen operaation ensimmäisen ja toisen parametrin. Esimerkiksi komento `diff 1 2` pyytää laskinta laskemaan lukujen 1 ja 2 erotuksen ja tulostamaan sen näytölle. `Div`-komennon tulos on liukuluku. Esimerkiksi komennon `div 3 4` tulos on 0,75. Ohjelma lukee käyttäjältä syötteitä, kunnes se lopetetaan parametrittomalla *quit*-komennolla.

Ohjelma tulostaa virheilmoituksen, jos komento on tuntematon tai tunnetulla komennolla on väärä määrä parametreja. Voit olettaa, että komento ja sen parametrit erotetaan toisistaan aina yhdellä välilyönnillä ja että syöte ei ala välilyönnillä tai lopu välilyöntiin. Oletetaan lisäksi, että aritmeettisten operaatioiden ensimmäinen ja toinen parametri ovat aina kokonaislukuja.

Tee ohjelmalle `String[]`-tyyppinen *tarkista*-metodi, joka tutkii onko parametrina saatu syöte (`String`) itsessään komento tai onko syötteen ensimmäinen osa komento ja onko komennolla oikea määrä parametreja. Paluuarvo on viite taulukkoon, jossa on komennon osat, kun syöte havaitaan oikeelliseksi. Jos parametriarvo on esimerkiksi "multi 2 2", niin taulukon sisältö on {"multi", "2", "2"}. Muodosta taulukko `String`-luokan `split`-metodilla ja käytä osien taulukkoa myös syötteen tarkistamisessa. Metodin paluuarvo on `null`, jos syöte on virheellinen tai `null`. Metodi ei lue tai tulosta mitään.

Kutsu metodiasi *Calculator*-ohjelman pääohjelmasta, jossa on pääsilmukka. Sijoita metodin palauttama arvo muuttujaan ja käytä muuttujaa, kun päättelet mitä tulostat näytölle. Tulosta pääohjelmassa virheilmoitus myös, kun jakaja (`div`-komennon toinen parametri) havaitaan virheelliseksi.

Esittele ohjelman tuntemat viisi komentoa *SUMMAA*-, *EROTA*-, *KERRO*-, *JAA*- ja *LOPETA*-nimisinä luokkavakioina ja käytä näitä vakioita metodissasi ja pääohjelmassa. Esimerkki *add*-komennon vakioinnista:

```
// Yhteenlaskun komento luokkavakiona.
private static final String SUMMAA = "add";
```

Esimerkki ohjelman toiminnasta:

```
Hello! I am a simple calculator.
Please, enter a command:
diff 1 2
-1
Please, enter a command:
add 10 3
13
Please, enter a command:
multi 2 2
4
Please, enter a command:
div 3 4
0.75
Please, enter a command:
add 1 2 3
Error!
Please, enter a command:
add 1
Error!
Please, enter a command:
add -1 -2
-3
Please, enter a command:
div
Error!
```

```
Please, enter a command:
div 0 3
0.0
Please, enter a command:
div 3 0
Error!
Please, enter a command:
quit now
Error!
Please, enter a command:
quit
```

3. Harjoitustyön infokalvoilla on esitelty harjoitustyössä käytettävien tekstitiedostojen sisältö. Kalvot löytyvät kurssisivujen *Harjoitustyö*-kohdasta.

   Kirjoita Javalla *lataaKuvaTaulukkoon*-metodi, joka luo kaksiulotteisen `char`-tyyppisten alkioiden taulukon ja lataa tekstitiedoston merkit taulukoon siten, että kuvan ensimmäisen merkki on taulukon ensimmäisessä paikassa (0, 0) ja viimeisen rivin viimeinen merkki on taulukon viimeisessä paikassa (n - 1, m - 1), missä n on taulukon rivien lukumäärä ja m on taulukon sarakkeiden lukumäärä.

   Luvut n ja m on annettu aina kuvatiedoston ensimmäisellä ja toisella rivillä, joita seuraavat tausta- ja edustavärien merkit omilla riveillään. Taulukkoon ladattavat kuvan merkit alkavat viidenneltä riviltä. Lue tiedostosta aluksi n ja m, jotta voit luoda $n \times m$ -kokoisen kuvan merkeille. Voit olettaa, että kuvan tiedot ovat kaikin osin kunnossa tässä tehtävässä, vaikka harjoitustyössä kuva voi olla joiltakin osin rikki.

   Tiedoston nimi välitetään metodille parametrina (`String`). Metodin toinen parametri on kahden merkin kokoinen yksiulotteinen taulukko (`char[]`), johon sijoitetaan tiedoston kolmannelta riviltä luettu taustamerkki (ensimmäinen alkio) ja neljänneltä riviltä luettu edustamerkki (toinen alkio).

   Metodin tyyppi on `char[][]`, koska metodi palauttaa viitteen merkit sisältävään taulukkoon. Paluuarvo on `null`, jos jälkimmäinen parametri on `null`-arvoinen tai siinä on väärä määrä merkkejä tai metodissa tapahtuu poikkeus esimerkiksi siksi, että tiedoston nimi on virheellinen. Huomaa, että metodisi ei lue tai tulosta mitään; vuorovaikutus käyttäjän kanssa tapahtuu muualla.

   Lue tiedoston nimi *FileLoader*-ohjelman pääohjelmassa ja kutsu metodiasi antamalla käyttäjän syöte ja pääohjelmassa luomasi kahden alkion kokoinen yksiulotteinen merkkitaulukko metodisi parametriarvoiksi. Sijoita paluuarvo muuttujaan, jonka tyyppi on `char[][]`, jotta a) metodisi luoma taulukko-olio ei häviä ja b) voit saada pisteen tästä tehtävästä.

   Sijoitus on erittäin tärkeä tämän tyyppisissä tehtävissä, koska tiedoston lataaminen kerran ja tuloksen myöhempi käyttö muuttujan avulla on huomattavasti tehokkaampaa kuin tiedoston lataaminen uudelleen eri kohdissa ohjelmaa. Tulosta kuvataulukko kutsumalla pääohjelmasta tehtävän 5.2 *tulosta2d*-metodia. Tulosta virheilmoitus, jos lataus epäonnistui.

Esimerkki, kun tiedoston *x.txt* rivit ovat "3", "5", "x", "o", "oxoxo", "ooxoo" ja "oxoxo":

```
Hello! I load files.
Please, enter file name:
x.txt
oxoxo
ooxoo
oxoxo
```

Esimerkki, kun hakemistossa ei ole tiedostoa *x-file.txt*:

```
Hello! I load files.
Please, enter file name:
x-file.txt
I could not load.
```

4. Kirjoita Javalla `boolean`-tyyppinen *tallennaRivit*-metodi, joka kirjoittaa parametrinaan saamansa kaksiulotteisen `char`-alkioista koostuvan taulukon merkit riveittäin tiedostoon, jonka nimi välitetään metodille sen jälkimmäisenä parametrina. Jokainen tiedoston rivi päätetään rivinvaihtoon. Jos taulukon merkit ovat esimerkiksi { { 'a', 'b' }, { 'c', 'd' }, {'e', 'f'} }, on tiedoston sisältö oheisen kuvan mukainen. Paluuarvo on `true`, jos tallentaminen onnistui. Paluuarvo on `false`, jos parametriarvoista jompikumpi tai molemmat ovat `null`-arvoisia, parametritaulukko on kooltaan pienempi kuin 1×1, nimiparametri on tyhjä merkkijono "" tai kirjoittamisen yhteydessä tapahtuu poikkeus. Metodi ei lue käyttäjältä syötteitä eikä tulosta näytölle.

```
ab
cd
ef
```

   Kutsu metodiasi vähintään viidesti *StringSaver*-ohjelman pääohjelmassa erilaisilla parametriarvoilla. Sijoita kunkin kutsun paluuarvo muuttujaan. Anna yhdessä kutsussa molempien parametrien arvoksi `null`.

5. Toteuta tehtävä 4.2 siten, että muutat *monista*-metodin rekursiiviseksi. Metodissa ei saa olla silmukoita. Älä muuta metodin otsikkoa. Esimerkiksi uuden parametrin lisääminen ei ole sallittua. Tämänkin ohjelman nimi on *Replicator*.

6. Potenssilaskussa eksponentti ilmaisee, montako kertaa luku kerrotaan itsellään. Esimerkiksi merkintä 2^3 tarkoittaa, että kantaluku kaksi kerrotaan itsellään kolmesti: $2 \cdot 2 \cdot 2 = 8$. Luvun nollannes potenssi on 1. Esimerkiksi $2^0 = 1$.

   Kirjoita Java-kielellä rekursiivinen *korotaPotenssiin*-metodi, joka laskee luvun potenssin, kun eksponentti on positiivinen (≥ 0) kokonaisluku. Metodissa ei saa olla silmukoita. Kantaluku ja eksponentti annetaan metodille `int`-tyyppisinä parametreina. Metodin paluuarvo on `double`-tyyppinen. Metodi palauttaa `Double`-luokan `NaN`-vakion, jos eksponentti on negatiivinen (< 0).

   Lue kantaluku ja eksponentti käyttäjältä *RecursiveExponentiator*-ohjelman pääohjelmassa ja kutsu metodiasi antamalla käyttäjän syötteet metodin parametrien arvoiksi. Sijoita metodin palauttama arvo muuttujaan ja kerro tämän muuttujan avulla tulos käyttäjälle. Voit tulostaa potenssilaskun tuloksen näytölle ilman nolladesimaalia esimerkiksi tyyppimuunnoksen avulla.

Käytä `Double`-luokan `isNaN`-metodia `Double.NaN`-arvon tunnistamiseen päämetodissa. Esimerkki:

```
...
// Kutsutaan metodia ja sijoitetaan tulos muuttujaan.
double potenssi = korotaPotenssiin(kantaluku, eksponentti);

// Eksponentti oli laillinen.
if (!Double.isNaN(potenssi)) {
...
```

Esimerkki ohjelman toiminnasta, kun syötteet ovat 2 ja 3:

```
Hello! I am a recursive exponentiator.
Please, enter base:
2
Please, enter exponent:
3
2 ** 3 = 8.
```

Esimerkki ohjelman toiminnasta, kun syötteet ovat 2 ja 0:

```
Hello! I am a recursive exponentiator.
Please, enter base:
2
Please, enter exponent:
0
2 ** 0 = 1.
```

Esimerkki ohjelman toiminnasta, kun syötteet ovat 3 ja -1:

```
Hello! I am a recursive exponentiator.
Please, enter base:
3
Please, enter exponent:
-1
Error!
```