1. rühmatöö

Rühmatöö põhiline ülesanne on koostöös kaaslasega kinnistada esimese viie praktikumi materjali ja seeläbi paremini valmistuda kontrolltööks. Silmas võiks pidada, et tuleb veel üks rühmatöö, milles võite (aga tingimata ei pea) 1. rühmatööd edasi arendada (eriti kasutajaliidese ja andmevahetuse osas).

Rühma suurus on 2 üliõpilast (erandina 3 üliõpilast), kes kõik on ühest praktikumirühmast. Kui on vajadus ja võimalus, siis tuleb muuta praktikumirühma. Rühmatöö saab üksinda teha ainult erandjuhtudel ja sellisel juhul on maksimaalne punktide arv kaks korda väiksem.

Arvestage rühmade loomisel sellega, et võib-olla tuleb nende inimestega natukese aja pärast omavahel jaotada miljoneid, kui see programm hakkab nii palju sisse tooma. (Kui koostöö ei suju, siis võib järgmises rühmatöös teisiti rühmituda. Vajadusel pöörduda praktikumijuhendaja poole.)

Hindamisel peab ülesande lahendust kaitsma juhuslikult valitud rühmaliige. Lahenduse hinne määratakse valemiga L*S, kus L on lahenduse korrektsus (kuni 5 punkti) ning S arv lõigust [0,5..1], mis näitab valitud rühma liikme selgituste adekvaatsust. Kõik kohalolevad rühmaliikmed saavad lahenduse eest tavaliselt võrdse hinde. Kui rühmaliikmete panus on väga erinev, siis võib hinne olla ka erinev.

Esitamine toimub R 13.03.2020 ja R 27.03.2020 praktikumi ajal!

Programm peab vastama järgmistele tingimustele.

- Programm käsitleb mingit (inimlikku) tegevust, näiteks mängimist, kliendile vastamist, kodumasina kasutamist vm. Eriti tore aga oleks, kui programm oleks teile kasulik oma erialaste ülesannete lahendamisel.
- Programm peab kasutajalt midagi küsima (võib-olla ka korduvalt). Võib eeldada, et kasutaja sisestab vastuse nõutud kujul.
- Programm peaks olema kasutatav ilma, et programmi kohta oleks erilisi eelteadmisi. Küsimused peavad vajaliku info andma. Hea oleks, kui programm käivitamisel annab vajaliku üldtutvustava lühiinfo.
- Programm peab sisaldama juhusliku suuruse kasutamist (soovitavalt klassi `Random` abil). Kui programm on mõeldud näiteks erialaste andmete töötlemiseks, siis võib sellest ka loobuda. Kuigi näiteks sobivas vahemikus juhuslike andmete genereerimine võib olla kasulik programmi testimiseks.
- Programm peab koosnema mitmest klassist (sh. peaklass). Andmete kasutamine peaks käima erinevate objektide abil. Selleks loodud klassid peaksid sisaldama isendivälju, konstruktoreid, `get`- ja `set`-meetodeid ja teisi vajalikke meetodeid.
- Programm peab olema rühmaliikmete endi kirjutatud.
- Programm peab olema mõistlikult kommenteeritud.
- Programm ei tohiks olla liiga keeruline. Vajadusel konsulteerige praktikumi juhendajaga enne programmeerimist.
- Kasutajaga suhtlemine peaks olema väga elementaarselt kujundatud. Vältida (veel) graafilist kasutajaliidest. Ekraanile kuvatav info (ka näiteks mänguseis) võib olla tekstiridadena (`System.out.println`). Kasutajaga dialoogi võib realiseerida mitmeti. Näiteks klassi `Scanner` (vt. `Praktikum 1`) kasutades või alltoodud `JOptionPane`

meetodeid kasutades. Programmi käivitamiseks vajalikku alginfot võib nõuda ka käsurea argumentidena (vt. Praktikum 2).
    - Kasutajaga dialoogi korraldamisel on abiks, kui programmi algusesse, lausa enne klassi kirjelduse algust, panna rida `import javax.swing.JOptionPane;`
    - Kohale, kus programm peaks kasutajalt sõne küsima, tuleb panna rida

```
String sisestatakse = JOptionPane.showInputDialog(null, "Sisesta midagi ",
"Andmete sisestamine",
                                                   JOptionPane.QUESTION_MESS
AGE);
```

Täpsemalt saate muidugi APIst vaadata!

Rühmatöö programmi arendamisel on väga soovitatav (eriti informaatika tudengitele) kasutada git versioonihaldustarkvara (näiteks GitHub vahendusel (juhend)).

Toome nüüd näiteks mõned võimalikud ülesanded. Väga soovitav on teil endil mingi teile huvitav ülesanne välja mõelda. Kuigi alltoodud nimekirjas on mitmed mängud, on soovitav, et ülesanne käsitleks hoopis midagi muud (näiteks erialaste andmete töötlust) või siis mäng oleks vähetuntud või suisa originaalne.

Ainult äärmisel inspiratsiooni puudumisel (mis ei ole tõenäoline) kasutage alltoodud näiteid.

- **Tikumäng**. Programmiga saab mängida mängu, kus laual on juhuslik arv tikku. Mängijad võtavad sealt kordamööda ära kas üks, kaks või kolm tikku. Kes võtab viimase tiku, on kaotaja.
- **Lohede võitlus**. Programm simuleerib kahe lohe omavahelist võitlemist. Klass Täring võiks olla teenusepakkuja.
- **Kalkulaator**. Koostada klassid nt täisarvude (`MyInteger`) ja reaalarvude (`MyDouble`) hoidmiseks (mis sest, et vastavad mähisklassid on olemas), milles oleksid meetodid lubatud tehete jaoks ja võrdlemiste jaoks. Testklassis küsida arvud ja tegevus, mida nende arvudega teha. Üks tegevus oleks juhuslik.
- **Mäng 15**. Kivid (arvud) 1 kuni 15 paigutatakse juhuslikult 4x4 ruudustikku. Kasutades tühja välja (tähistada andmetes 0-ga) liigutada kive, nii et tekiks õige järjestus. Liigutada saab vaid tühja koha naabrit - kivi, mis on tühjast kohast vasakul, paremal, üleval või all (max 4 naabrit).
- **Täringumäng**. Koostada klassid `Täring` ja `Täringumäng` jäljendamaks kahe mängija mängu. Mängijate viskevoorud kordamööda. Ühes viskevoorus võib mängija visata täringuid ükskõik mitu korda, aga kui tuleb 1, siis kogusumma nullitakse ja viskeõigus läheb teisele mängijale. Muidu viskel saadud tulemus liidetakse kogusummale. Võidab see, kes saab enne üle 91 punkti.
- **Trips-traps-trull**. Mängu alustaja valitakse juhuslikult. Ruudustik on nummerdatud numbritega 1-9 (näiteks nagu mobiiltelefoni klaviatuur). Käigu tegemiseks tuleb sisestada ruudu number. Mängulaud joonistada kriipsudega konsoolile (ning iga käigu järel väljastatakse uus ruudustik).
- **Bussipiletide ostmine**. Kasutajalt küsitakse, mitut piletit ta vajab ning pakutakse talle siis vastav arv juhuslikke, mis pole veel ära ostetud. Kui kasutaja on nõus need ostma, siis need müüakse ja järgmisele kasutajale enam neid ei pakuta. Programmi töö lõpeb, kui (järjekordne) kasutaja küsib rohkem pileteid kui vabu kohti jäänud on.