Eksamiküsimuste vastused aines Tarkvaratehnika (MTAT.03.094)
Aeg: 09. jaanuar 2007, 09:00 – 13:00

Küsimused

1. (5 p.)
   Küsimus:
   Kirjelda võimalikult täpselt klassiskeemi ülesehitust (klassid, seosed, jne) enda toodud näite varal.

   Vastus:
   Kirjeldatud ning näites esindatud peavad olema:
   - Pakett (0.5 p.)
   - Klass (0.5 p.)
   - Liides (0.5 p.)
   - Atribuut (nimi, tüüp) (0.5 p.)
   - Meetod (nimi, parameetrid, tüübid) (0.5 p.)
   - Nähtavus (public, private, protected, package) (0.5 p.)
   - Seos (mitmesus, rollid) (0.5 p.)
   - Agregeerimine, koostamine (0.5 p.)
   - Pärimine (0.5 p.)
   - Sõltuvusseos (0.5 p.)

2. (5 p.)
   Küsimus:
   Loetle väleda tarkvaraarenduse ideaalse keskkonna tingimused. Mis tähendust omab arendusprotsessi korraldamise mõttes arenduse ideaalne keskkond? Kirjelda iga tingimuse korral, mida arendusprotsessis tuleb korraldada, kui antud tingimus ei ole täidetud.

   Vastus:
   - Ideaalses arenduskeskkonnas ei ole vaja tegeleda arendusprotsessi korraldamisega. (0.1p.)
   - Tingimused:
     - "*2..8 inimest samas ruumis*" - kui inimesi on rohkem, või mitte ühes ruumis, siis tuleb tarvitusele võtta täiendavad meetmed (dokumentatsioon, koosolekud, ...), et toimuks konstruktiivne suhtlus. (0.7 p.)
     - "*valdkonna spetsialist samas ruumis*" - kui valdkonna spetsialist / klient ei saa täiskoormusega arendajatega koos olla, on vaja tagada, et arendajate ja kliendi vaheline suhtlus toimiks. (0.7 p.)
     - "*1-kuulised inkremendid*" - kui tehakse poole aasta pikkust waterfall'i, on vaja vahepealseid nõuete spetsifikatsioone ja disainidokumente ning neid hoolikamalt verifitseerida, et tagada, et hiljem vähem ümbertegemist oleks. (0.7 p.)
     - "*Täisautomaatsed regressioonitestid*" - kui ei saa kogu testimist automatiseerida, tuleb käsitsi testida. Kuna käsitsi testimine on mahukas töö, tuleb seda efektiivseks toimimiseks hoolikalt plaanida ja seirata (monitoorida). (0.7 p.)
     - "*Kogenud arendajad*" -- kui tiimis on vähekogenud arendajaid, tuleb eraldi pingutada selleks, et nende kogemuste puudujäägid katta (nt. arhitektuuri patterneid rohkem lahti joonistada, dokumenteerida töövõtteid, kontrollida hoolikamalt antud arendajate tulemite kvaliteeti). (0.7 p.)
     - "*Toimiv konfiguratsioonihaldus*" - juurutada toimiv

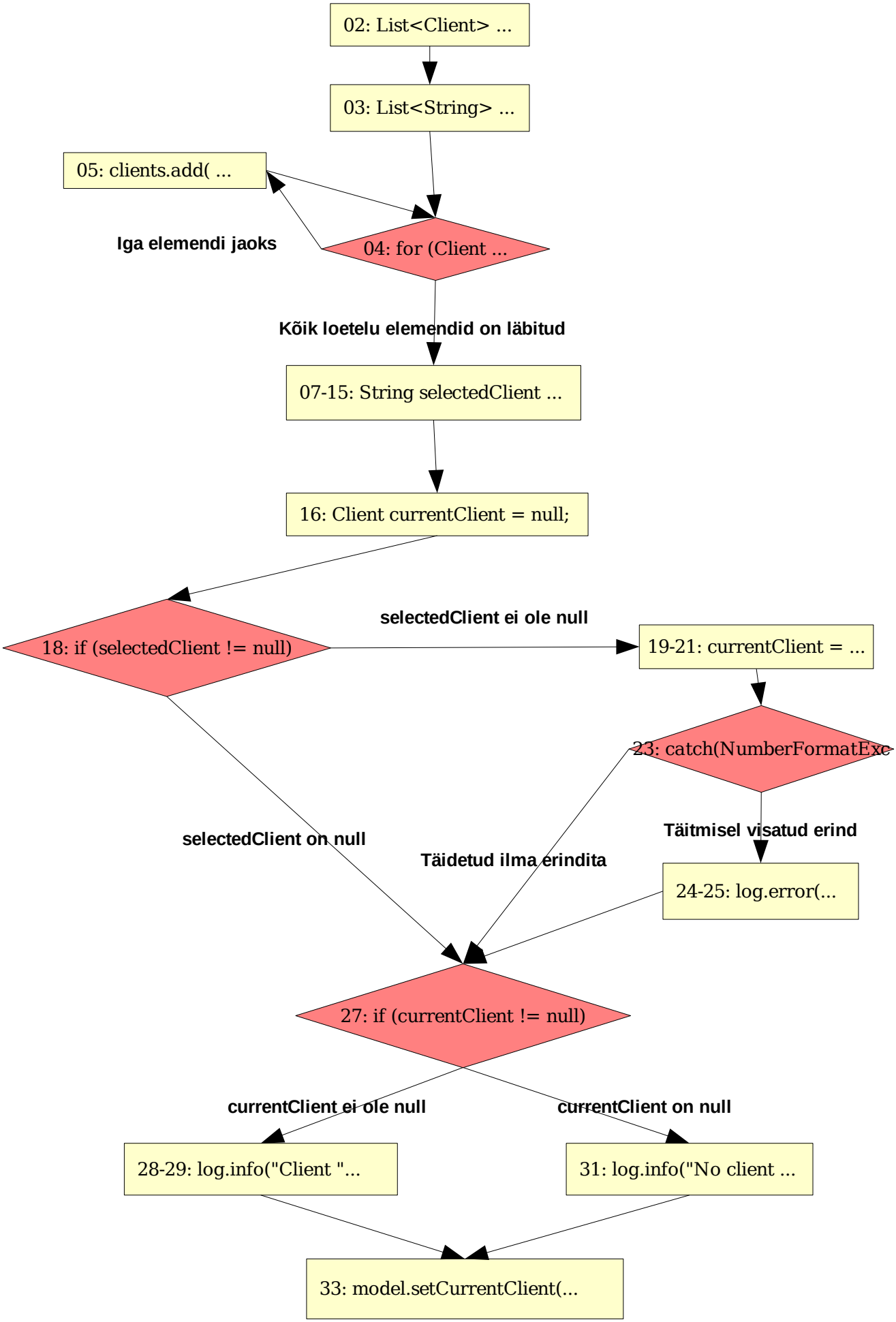
konfiguratsioonihaldus. ilma selleta pole midagi teha. (0.7 p.)
    - "*Puuduvad olulised pudelikaelad*" - Kui protsessis on oluline pudeli kael, kuhjuvad tööd sinna ja mitte-pudelikaelaks olevad lülid hakkavad oma tööde läbi surumiseks täiendavalt üle kuhjama. Sundida mittepudelikaelas olevaid lülisid oma sisendit hoolikamalt ette valmistama pudeli kaela jaoks. Nt. on tiimis 6 UI developeri ja 1 andmebaasi disainer, kes on pudelikael. UI developerid parem tehku paar ekstra iteratsiooni ilma andmebaasi spetsialistita oma päringuspetsifikatsioonide kvaliteedi tõstmiseks hilisemate paranduste mahu vähendamiseks -- see võib UI developeri üksikindiviidi seisukohalt ebaefektiivsust suurendada, aga systeemi kui terviku efektiivsust see tõstab. (0.7 p.)

3. (5 p.)
   Küsimus:
   Arvuta antud koodilõigu (vt järgmine lehekülg) *McCabe's Cyclomatic Complexity* meetrika väärtuse:
   - Joonista koodilõigu täitmise voo diagrammi.
   - Seleta kuidas väärtus oli saadud: millise valemi järgi, millised on valemi muutujate väärtused, kuidas muutujate väärtused olid arvutatud.

   Hindamine
   - Tätimise voo skeem (3 p.)
   - Arvutus (2 p.)

   Lahendus:
   - McCabe's Cyclomatic Complexity Definitsioon: Meetrika mis näitab lineaarselt sõltumatu teede arv programmi lähtekoodis
   - Meetrika arvutamiseks kasutatakse programmi täitmise skeemi. See skeem on suunatud graaf, mille tipud on programmi avaldised ja kaar ühendab 2 tipu kui võib juhtuda, et teine avaldis täidetakse kohe peale esimest.
   - Teisisõnu näitab see meetrika mitu hargnemist on programmi täitmise graafis.
   - Meetrika väärtuse saamiseks on olemas mitu valemit. Loengu jooksul oli juttu kahest:
     - V(G) = e – n + 2, kus
       - e on kaarte arv graafis
       - n on tippude arv graafis
     - V(G) = d + 1, kus
       - d on kontrollavaldiste arv programmis (if, while, for, ...)
   - Koodilõigu täitmise skeem: vt. allpool. Iga koodirida on nummerdatud ja täitmise skeemis kasutatakse see numeratsioon.
   - Koodilõigu V(G) arvutus
     - Valemi V(G) = e – n + 2 järgi
       - täitmiseskeemis on
         - 17 kaart
         - 14 tippu
         - järelikult V(G) väärtus on 17-14+2=3+2=5
     - Valemi V(G) = d + 1 järgi
       - koodis on 4 kontrollavaldist (skeemi peal need on tumedamad): 1 for, 2 if ja 1 catch, järelikult V(G) väärtuseks on 4+1=5

```
01 private void drawSelectClientDialog() {
02  List<Client> allClients = domainController.loadAllClients();
03  List<String> clients = new ArrayList<String>();
04  for (Client client: allClients) {
05    clients.add(client.getId() + ". " + client.getFirstName());
06  }
07  String selectedClient =
08    (String)JOptionPane.showInputDialog(
09      this,
10      Translations.getString("main.chooseCustomer"),
11      Translations.getString("main.chooseCustomer"),
12      JOptionPane.OK_CANCEL_OPTION,
13      null,
14      clients.toArray(),
15      0);
16  Client currentClient = null;
17  try {
18    if (selectedClient != null) {
19      currentClient =  domainController.getClientById(
20          Long.parseLong(selectedClient.
21            split(" ")[0].replaceAll("\\.","")));
22    }
23  } catch(NumberFormatException e) {
24      log.error("Failed to parse client id," +
25        " probably no client was selected");
26  }
27  if (currentClient != null) {
28    log.info("Client " + currentClient.getFirstName() +
29      " with ID=" + currentClient.getId() + " got selected.");
30  } else {
31    log.info("No client selected");
32  }
33  model.setCurrentClient(currentClient);
34 }
```


02: List<Client> ...
03: List<String> ...
05: clients.add(...
Iga elemendi jaoks
04: for (Client ...
Kõik loetelu elemendid on läbitud
07-15: String selectedClient ...
16: Client currentClient = null;
18: if (selectedClient != null)
selectedClient ei ole null
19-21: currentClient = ...
23: catch(NumberFormatExc
selectedClient on null
Täidetud ilma erindita
Täitmisel visatud erind
24-25: log.error(...
27: if (currentClient != null)
currentClient ei ole null
currentClient on null
28-29: log.info("Client "...
31: log.info("No client ...
33: model.setCurrentClient(...

4. (10 p.)
   Ülesanne:
   Valdkonna kirjeldus.

   Ehitusbuumi tingimustes vajab uut tarkvara ehitusseadmete laenutusega tegelev ettevõte. Valdkond on üsna sarnane nt. videolaenutusega, kuid laenutuse hind varieerub laenutuse pikkusest (tund, pool päeva, päev, nädal, …). Ettevõte müüb lisaks seadmetes kasutatavaid materjale, nt. liivapaberit lihvimismasina jaoks. Ettevõte müüb erijuhtudel ka renditavaid seadmeid (laenutaja hoiab seadet kauem kui maksimaalne rendiperiood või kahjustab seadet nii et seda ei saa enam välja rentida). Ettevõte kogub trahve vahenditelt mis on tagastades kahjustatud kuid parandatavad. Seadme parandamistööd tellitakse välistelt partneritelt, võimaluse korral kasutatakse garantiiremonti.

   Ülesanded:
   - Leia tegutsejad ning nende eesmärgid antud tegevuse kontekstis.
   - Koosta kasutuslood „Rendi seade” ning „Tagasta seade” koos võimalike eel- ja järeltingimus(t)e ning kõrvalstsenaariumitega.
   - Leia ja kirjelda lühidalt veel kaks kasutussituatsiooni kasutades ainult põhivoogu.

   Lahendus & hindamine:
   - Tegutsejad: rendileandja, rendilevõtja, remonditöökoda. (1 p.)
   - Kasutuslood „Rendi seade” ning „Tagasta seade” (4+4 p.):
     - Esimeses järjekorras käsitleda loovust ning põhjalikkust.
     - Sisse peab olema toodud lepingu või selle sünonüümi mõiste ning olema näha lepingu olekute käsitlus (leping sõlmitud, leping lõpetatud, leping muudetud, lepingule loodud lisa seadme lõhkumise korral, …).
     - Selgelt ja loetavalt peavad olema välja toodud kõrvalstsenaariumid. Põhistsenaarium ei tohi nende sisse ära kaduda. Üleminekud põhi- ja kõrvalstsenaariumitest peavad olema selgelt arusaadavad. Näha peavad olema erinevad lõppolekud (leping lõpetatud, leping muudetud, …)
     - Väliste partnerite osalemise korral peavad olema käsitletud tõrkesituatsioonid.
   - Kasutuslood: seadme ost, seadme müük, seadme remont, … Kasutuslugude nimetamise eest võib anda kuni veerandi punktidest. Ülejäänu analoogiliselt teises punktis toodule. (1 p.)

5. (10 p.)
   Ülesanne:
   Koosta lihtsa tabelarvutusprogrammi (a la *Microsoft Excel*, *OpenOffice Calc*) disain.

   Funktsionaalsus:
   - Tabeli suurus ei ole piiratud.
   - Tabelis saavad olla: arvud, tekstid, valemid.
   - Valemiteks on: lihtne aritmeetika `[ + - * / ]`, vahemiku summa `[SUM(A1:A20)]`, astendamine `[A1^3]`, ruutjuur `[SQRT(2)]`.
   - Valemites võivad sisalduda nii konkreetsed arvud kui ka viited teiste lahtrite väärtustele.
   - Kui ühe lahtri sisu muutub, siis muutuvad automaatselt kõik sellest sõltuvad lahtrid ja omakorda neist sõltuvad jne.

- Tabeli salvestamine ja laadimine.
- Tabelis saab ringi liikuda kasutades nii nooleklahve kui ka kerimisribasid tabeli servades.

Vastus peaks sisaldama:
- klassiskeem,
- käitumist kirjeldavad skeemid,
- selgitav tekst,
- olulisemate meetodite pseudokood,
- olulised erijuhud, veasituatsioonid,
- kasutatud disaini mustrid.

Swing (või mõne teise sarnase) API tundmine pole oluline, tähtis on põhimõte, kasutada võib pseudokoodi.

Igasuguse disaini juures on oluline, et tulemus oleks selgelt arusaadav, kood hästi loetav ning hilisem muudatuste/täienduste tegemine lihtne.

Mõtle, kui lihtne oleks sellise disaininga hiljem realiseerida selliseid täiendusi:
- iga üksiku lahtri või lahtrite grupi formateerimine (teine shrift, rasvane kiri, joondamine jne),
- *copy* & *paste* tugi,
- tabeli (osa) trükkimine,
- uute valemite (funktsioonide lisamine).

Hea disaini korral peaks nende funktsioonide lisamine tähendama minimaalseid muutusi olemasolevas koodis ja lihtsalt uue loogika juurde kirjutamist.

Lahendus:
- Mõistlik oleks, kui on eristatav midagi MVC sarnast
  - Model: tabel ja selles sisalduvad lahtrid, nende omavahelised seosed
  - View: vaade tabeli osale
  - Controller: sisendi vastuvõtt ja edastamine õigele mudeli osale
- Tabeli disainis on oluline, kuidas tabelit mälus hoitakse. Oluline on, et tabeli iga lahtri kohta ei tehta ette objekti, kui seal midagi pole. Üks parimaid variante on HashMap või midagi sarnast, kus on sees ja kiirelt kätte saadavad täidetud lahtrid aadressi järgi (A1, B3 jne.)
- Ühe lahtri disaini võib teha mitmeti:
  - võib olla kirjeldatud interface'na (või abstract class'na), millest on erinevaid implementatsioonid (tekst, number, valem).
  - võib olla üks klass, millel on erinevad strateegiad, kuidas tema sisu esitada.
- Lahtri disainis peab olema eristatav selle väärtus ja sinna sisestatud tekst (oluline just valemite korral). Hea oleks kui tudeng on mõelnud, kuidas käituda, kui valem on vigane.
- Iga lahter võib mõjutada teisi lahtreid, mis seda valemites kasutavad. Seega on oluline, et lahtril oleks küljes nimekiri temast sõltuvatest lahtritest, kellele antakse signaal, kui lahtri väärtus muutub.
- Eraldi oluline osa on valemite parsimine ja parsitud kujul mälus hoidmine. Parsimise käigus peaks leitama aluseks olevad lahtrid ning nendele end sõltuvaks registreerima (Listener).
- Hea oleks, kui on mõeldud nendele aspektidele:

    - Kuidas käitutakse valemite muutmisel, kui sõltuvused muutuvad.
    - Kuidas tuvastatakse ringsõltuvuse tekkimine ning mida siis tehakse.
- Kuna valemid võivad olla mitmeosalised, siis nende hoidmine võiks ideaalis olla mingis Composite taolises hierarhias. Composite'il oleks meetod a la getValue(), mis siis sõltuvalt osa tüübist annab enda väärtuse või küsib enda all olevate osade getValue() ning sooritab mingi operatsiooni vastustega ning tagastab tulemuse.
- Funktsioonidele vastavate objektide (eelmainitud Composite osad) loomiseks oleks mõistlik kasutada Factory't, mis omaks nimekirja kõigist võimalikest funktsioonidest.
- Salvestamise/laadimise jaoks peab olema tabeli faili serialiseerimise/deserialiseerimise loogika paigas.
- View osas peab olema arvepidamine selle üle, milline osa kogu tabelist on hetkel nähtav. Kirjeldatud peaks ka olema, kuidas toimub nähtava osa joonistamine - kuidas käiakse läbi read-veerud ning kuidas saadakse lahtri väärtus. Lisaks peab olema kirjeldatud kuidas toimub lahtri sisu redigeerimine.
- Controlleris peab olema eristamine fookuse liigutamise ja väärtuste/valemite sisestamise vahel. Esimesel juhul tuleb muuta view'd, teisel juhul edastada sisestatu mudelile. Lisaks ka reageerimine save/load soovidele.

Hindamine:
- Vastus peab pakkuma lahendused ülaltoodud teemadele ning sisaldama ka mingil kujul klassidiagrammi.
- Oluline on lahenduse selgus ning lihtsus ja erinevate alamosade eristatavus.
- Standardite (nt UML) või APIde (nt Swing) täpne tundmine ei saa olla määrav.