

Nuottipiirturi - Tekninen suunnitelma

kurssille Ohjelmointistudio2 (MOOC)

Niina Saarelainen

ei opiskelijanumeroa

(koulutus: pianonsoitonopettaja ja musiikkiteknologian maisteri)

10.2.2017

1. Ohjelman rakennesuunnitelma

Seuraavalla sivulla on ohjelman UML-kaavio ilman metodikuvauksia.

Luokka *NuottiPiirturi* on ohjelman logiikkakeskus. Se on yhteydessä kaikkiin muihin luokkiin ja niiden olioihin. *NuottiPiirturissa* tapahtuu nuottidatan käsittely ja sen pohjalta ohjeiden antaminen muille luokille ja metodeille. Tärkeimpiä metodeita ovat tunnisteidenKasittely, lyriikanKasittely ja nuottidatanKasittely, joka sisältää nuotti- ja tauko-olioiden luonnin ja lisäämisen ArrayBufferiin.

Kaikki nuotit, soinnut ja tauot periytyvät *ViivastolleLaitettava* -piirreluokasta. Yhteistä kaikille on kuva, jota piirretään käyttäen perittyjä ominaisuuksia, esim. samanlaista nuotin nuppia. Esimerkiksi korkeus ei ole yhteinen piirre, koska soinnulle ei voi määritellä vain yhtä korkeutta. Abstrakteilla Nuotti- ja Tauko-luokilla on ominaisuudet pituus ja korkeus. Kaikki nuotit ja tauot perivät nämä ominaisuudet.

Sointu¹ on kokoelma Nuotteja ja Kahdeksasosapari muodostuu graafisesti ennemminkin kahdesta 1/4-nuotista kuin kahdesta 1/8-nuotista (ei haluta yksittäisiä väkäsiä kahdeksasosapariin).

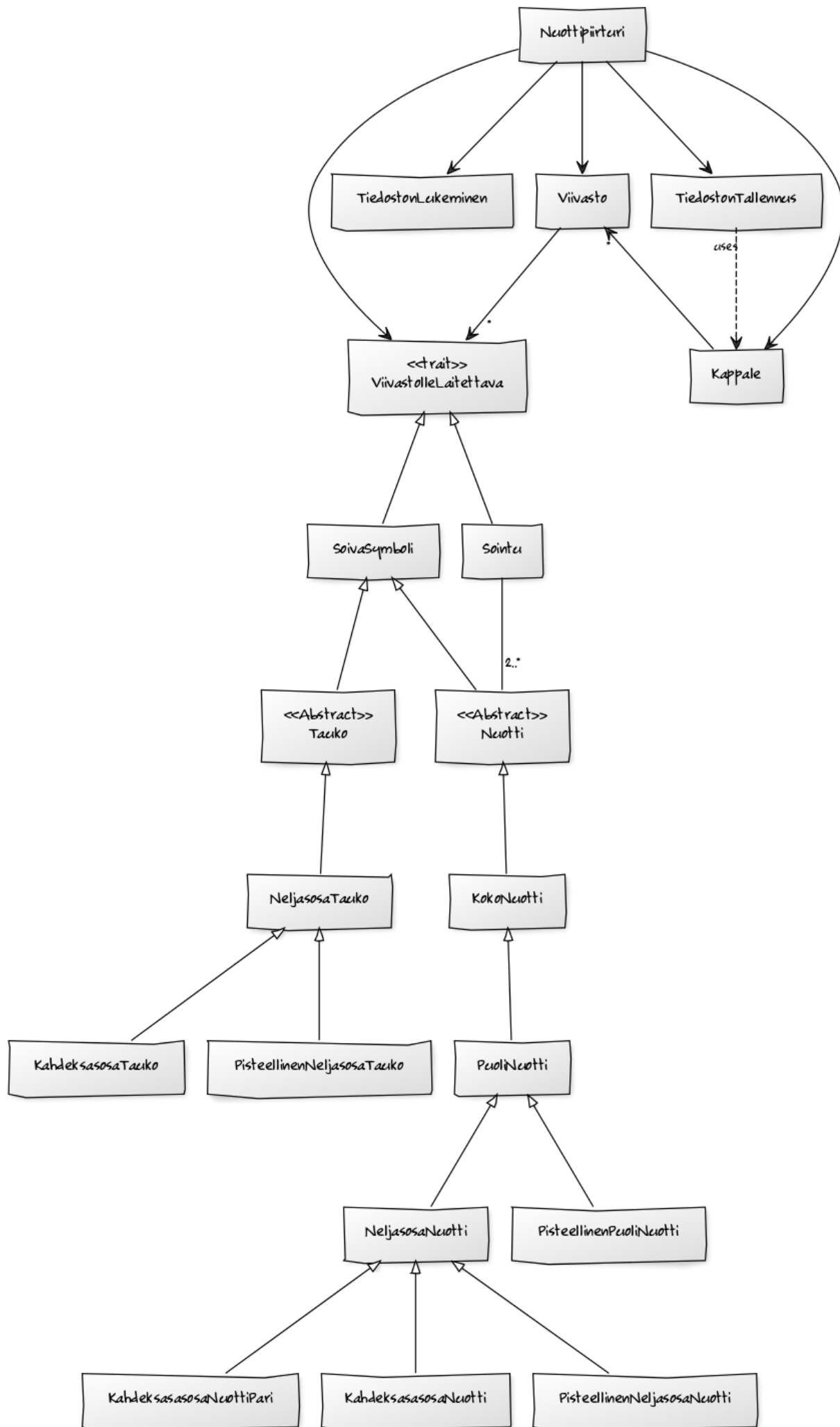
Luokan *TiedostonLukeminen* olio lukee tiedostosta nuottidatan. Tarkempi datan käsittely tapahtuu *NuottiPiirturi:ssa*

TiedostonTallennus-olio tallentaa tiedoston käyttäjän haluamalla nimellä.

Kappale-olio sisältää Viivasto-olioita, joita laitellaan talteen kun yksi viivasto on saatu täyteen.

Viivasto piirtää *ViivastolleLaitettava*- olioiden kuvakenttiä ja jos kappaleessa on sanat, ne kirjoitetaan synkroonissa kuvien kanssa

¹ Aion kutsua interallejakin soinnuiksi. Sointu on siis > 1 nuotti samanaikaisesti soitettuna. Tässä toteutuksessa kaikkien soinnun sävelten tulee olla saman mittaisia. Ohjelma ei kaadu jos jonkun tekee vahingossa liian pitkäksi, mutta tahtiviivojen laittaminen oikealle paikalle muuttuu mahdottomaksi tämän jälkeen.



2. Käyttötapauskuvaus

Käyttötapaus #1: Halutaan nuotit valmiista syötteestä

- 1) Käynnistyksen jälkeen käyttäjää pyydetään kirjoittamaan konsoliin tiedoston nimi. TiedostonLukeminen-olio lukee tiedon, ja NuottiPiirturi tallettaa sen yksittäisiksi alkioiksi (nuoteiksi, tauoiksi, sointu) eriteltynä.
- 2) Tietoa aletaan käsitellä alkio kerrallaan. Jokaisesta alkioista muodostetaan sitä vastaava ilmentymä. Alkioita vastaavat ilmentymät ovat tyyppiä nuotti, tauko tai sointu, kaikki samaa yläluokkaa, joten ne voi tallettaa samaan puskuriin.
- 3) Viivasto hoitaa piirtämiseen liittyvät operaatiot. Viivasto piirtää nuotteja ViivastolleLaitettava-datapuskurista, laskee milloin on tahtiviivan aika ja mahdollisesti lukee synkronissa lyriikkapuskuria, jos syötteessä oli sanat.
- 4) Kun työn alla oleva viivasto on täynnä, lisätään se Kappale-luokan ilmentymään.
- 5) Kun koko ViivastolleLaitettava-datapuskurin sisältö on käsitelty eli kappale on loppu, tallettuu kappale TiedostonTallennus-luokan olion avustuksella tiedostoon.
- 6) Tässä vaiheessa tulee konsolille teksti: "Millä nimellä tallennetaan?"

Käyttötapaus #2: Käyttäjä haluaa ensin tehdä syötekappaleen:

- 1) Käynnistyksen jälkeen konsollilla on ohjeet syötedatan, eli nuoteiksi haluttavan kappaleen tekemiseksi. Taulukon ja esimerkkien muodossa selitetään, miten merkitään eri mittaiset ja korkuiset nuotit, tauot ja soinnut. Ja mahdolliset sanat.
- 2) Käyttäjä tekee tekstitiedoston, jonka sisältö on: "c1 c2 g a1" ja tallettaa sen oikeaan paikkaan.
- 3) Käyttäjä valitsee tekemänsä tiedoston kirjoittamalla sen nimen konsolille
- 4) Tulee virheilmoitus: "Tarkenna nuotti g rivillä 1: g1 vai g2? Korjaa tiedostoon tai valitse toinen tiedosto."
- 5) Käyttäjä mahdollisesti editoi tiedostoa, ja valitsee listauksesta jonkin tiedoston.
- 6) Loppu ohjelma toimii kuten käyttötapaus #1:ssä.

3. Algoritmit

Syötedata ("g2- z- <f2-,g1-> z d2 <f2-,g1-> z d2 f2"), josta ensin poistetaan tyhjät rivit ja liiat välilyönnit, sijoitetaan rivi kerrallaan puskuriin. Sitten käsitellään mahdolliset tunnisteet nimi, tahtilaji, sanat ja vapaaehtoiset tunnisteet (esim. kappaleen rakennetta selventävät merkinnät, kuten #C-osa). Yksikään tunniste ei ole pakollinen: jos nimi puuttuu, kappaleelle ei tule otsikkoa. Jos tahtilaji-tunniste puuttuu, tehdään kappale 4/4.

Loppu data on nuottidataa, joka tallennetaan omaan puskuriin, josta se jatkokäsitellään nuotti/tauko/sointutapahtuman mittaisiksi alkioiksi splittaamalla välilyönnistä. Tätä taulukkoa luetaan alkio kerrallaan, jolloin luodaan oikeanmittainen ja -korkuinen nuottiolio ja tallennetaan se puskuriin. Nuotin pituus selviää merkin "-" (yksi isku) lukumäärästä, pisteellinen nuotti sisältää ".". Jos näistä ei ole kumpaakaan, on nuotti, tauko tai sointu kahdeksasosan mittainen.

Nuotin korkeus on yleensä se minkä käyttäjä on kirjoittanut. Kuitenkin etumerkkilogiikka (#, b, §) vaatii pitämään tahdin ajan muistissa mikä/mitkä äänet on jo ylennetty. Oikeassa nuottikirjoituksessahan etumerkit ovat voimassa tahtiviivaan asti ja niitä on tarpeeton piirtää uudestaan. Toteutan saman ylläpitämällä puskuria, johon laitetaan tahdin ajaksi kaikki ylennettyjen tai alennettujen nuottien nimet. Sitä pitää tutkia jokaisen nuotin piirtohetken kohdalla, myös

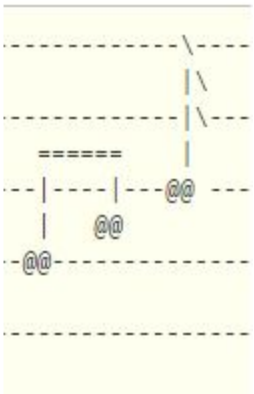
ylentämättömien, koska jos nuottidata on: "c#1 c1", tulee c1:n kohdalle palautusmerkki vaikka se ei syötemuodossa ilmene mitenkään.

Kahdeksasosaparin löytäminen datasta: Nuoteissa kaksi perättäistä kahdeksasosaa laitetaan samaan palkkiin, jos ensimmäinen on iskulla. Kuitenkaan kaksi kahdeksasosaa jotka ovat alleviivatuilla tahtiosilla, ei laiteta samaan palkkiin : Yk-si kak-si.

Eli lukiessa tulee pitää kirjaa millä tahdinosalla mennään ja lukea kahdeksasosan ja soinnun (tunnistetaan "<>"-merkeistä) tapauksessa mahdollisesti myös seuraava nuotti puskurista, jotta osataan antaa käskyn oikeanlaisen nuottiolion luomisesta.

Soinnun tapauksessa luodaan ensin sen sisältämät nuottioliot, jotka sitten toimitetaan uudelle Sointu-oliolle jatkokäsittelyyn. Siellä piirtämiselle pitää antaa lisäohjeita. Esimerkiksi jos on laitettava kolme yksittäistä kahdeksasosaa väkäsineen päällekkäin, näyttää tulos lähinnä 1/32-nuotilta, jos vain annetaan ohje piirtää 3 yksittäistä kahdeksasosaa. Samoin kahdeksasosan palkki pitää piirtää vasta lopuksi.

Grafiikan piirtäminen tapahtuu ViivastolleLaitettava-luokan perivien luokkien instansseissa. Nuotti-/sointu- /tauko-olioissa piirretään kuva itsestään ja ympäröivästä viivastosta, kuva sivulla 6. Myöhemmin näitä nuotti+viivastomerkkijonoja vain liitetään peräkkäin, kun Buffer[ViivastolleLaitettava]-tyyppistä oliota luetaan.

	<p>Nuotin kuva ei ole aina samanlainen. Joka nuotilla kokonuottia lukuunottamatta on kaksi mahdollista varren suuntaa. (kuvassa vain ylöspäin) . Nuotit c1-h1 haluavat varren ylöspäin, loput alaspäin.</p> <p>Lisäksi kahdeksasosaparin tapauksessa nuottien varren pituus ei ole vakio, kuten projektisivun https://plus.cs.hut.fi/studio_2/2017/studioprojekti/283/ kuvasta voi havaita. Varren pituuteen vaikuttaa se kuinka kaukana parin toinen nuotti on ensimmäisestä. Parin tapauksessa tulee siis aina tutkia kahden nuotin ominaisuuksia ennen kuin varren suunnan ja pituuden voi piirtää. Kahdeksasosaparin tapauksessa lasken kummalla puolen on enemmän tilaa, siis on verrattava kumpaakin nuottia suhteessa piirtoalueen reunoihin. Tässä siis toisen varsi saattaa piirtyä vastoin edellä esitettyä varsisääntöä.</p>
---	--

Lopullista viivastoa muodostettaessa laitetaan rivin alkuun G-avain, sitten peräkkäin ViivastolleLaitettava-luokan instanssien tuotoksien kuva-kenttiä. Sanoitukselle on paikka viivaston alimman nuotin alapuolella. Sanoja kirjoitetaan tavu kerrallaan nuotti- ja sointutapahtumien kanssa. Tauoilla ei lauleta.

Tahteja laitan neljä samalle viivastolle, joka määrä on länsimaisen musiikin yleisin fraasinpituus. Tahtiviiva pitää laittaa kun Nuotti- tai Sointu-oliosta saatu pituuskentän arvo lisättynä saman tahdin saldoon == tahtilaji-tunnisteen luku tai oletusarvo 4.

Tallennettava tiedosto koostuu kappaleen nimestä, joka saadaan tunniste-puskurista sekä Kappale-luokan ilmentymästä.

4. Tietorakenteet

4.1 luokka TiedostonLukeminen

Syötetiedosto saattaa olla esimerkiksi seuraavanlainen:

```
#Nimi Ukko Nooa  
#4    c1- c1- c1- e1- d1- d1- d1- f1- e1- e1- d1- d1- c1----  
#sanat Uk-ko Noo-a Uk-ko Noo-a o-li kun-non mies
```

Yllä oleva tieto tallennetaan Source.fromFile:ia käyttäen Buffer[Stringiin]. Tähän rakenteeseen on helpoin lisätä, koska rivi kerrallaan tutkitaan ennen lisäämistä kannattaako "tietoa" lisätä (ei tyhjiä rivejä yms), lisäksi syötteen määrälle ei ole rajoitteita, joten vakiokokoista taulukkoa ei missään nimessä kannata mennä luomaan.

4.2 luokka NuottiPiirturi

Käytössä on useita ArrayBuffer:ita säilömässä tietoa, ovat tyyppiä [String] tai [ViivastolleLaitettava]. Pari Array[String]:iäkin muodostuu split-operaation tuloksena. Lisäksi luokka luo ilmentymiä kaikista muista luokista tarvittaessa.

Laajennettavuuden näkökulmasta Buffer[ViivastolleLaitettava] on erityisen kätevä: Puskurissa olevasta nuottitiedosta voi kohtuullisen helposti (ehkä?!) koodata parempaa grafiikkaa käyttävän version. Lisäksi hienoja lisäpiirteitä olisivat kappaleen soittaminen MIDI:ä käyttäen ja nuottiviivastojen automaattinen skrollaaminen musiikin tahdissa. Nämä jäänevät tämän toteutuksen ulkopuolelle, mutta mahdollistuvat, kun nuottidata on tallessa tietorakenteessa, versus se että nuotit vain piirtää yksitellen ja mitään ei jää talteen. Puskuri mahdollistaisi myös transponoinnin.

4.3 trait ViivastolleLaitettava ja sen perivät luokat

Piirteen nimi ja ominaisuudet on valittu laajennettavuutta silmälläpitäen. Myöhemmin (tämän kurssin jälkeen) saatan innostua laajentamaan ohjelmaa. Tällöin on hyvä että ohjelma osaa tallentaa nuotit ja tauot puskuriiin, joka hyväksyy mahdollisimman "yleistä" tyyppiä. Eli ViivastolleLaitettavan aliluokiksi voisi keksiä vaikkapa Artikulaatiomerkin tai Sointumerkin.

Takaisin maanpinnalle: Map["nuotin nimi" -> Int] kertoo missä viivaston y-akselikohdassa mikin nuotti sijaitsee. Abstraktit luokat Nuotti ja Tauko tuovat nuoteille ominaisuudet korkeus ja pituus. Pituus-tietoa tarvitaan vielä myöhemmin ohjelmassa, kun lasketaan tahtiviivojen paikat.

4.4 luokka Viivasto

Viivasto on Buffer[String]

4.5 luokka Kappale

Kappale on Buffer[Viivasto]

4.6 luokka TiedostonTallennus

ReadLinella kommunikoidaan käyttäjän kanssa ja PrintWriter tallettaa Kappale-olion Viivasto-olioita rivi kerrallaan.

5. Aikataulu

Helmikuuun aikana saadaan toimimaan nuottidatan käsittely, esim että soinnut tunnistuvat. Nuottien piirtäminen aloitetaan helpoimmista ja etumerkkien kanssa ei vielä keulita. Maaliskuussa mietitään tarkemmin kahdeksasosien ja sointujen piirtämistä. Lyriikat tulevat vasta tässä vaiheessa mukaan. Heti kun jonkinlainen toimiva versio on saatu aikaan, voin kysyä muusikkokaverilta mitä hän pitää käyttöliittymästä ja syötetiedoston muodosta. Huhtikuussa testataan, parannellaan koodia ja saadaan homma valmiiksi.

Ehdin tehdä noin 12h viikossa, eli 14 viikossa käytän tähän projektiin noin 168 tuntia.

6. Yksikkötestaussuunnitelma

Viitaten yleissuunnitelmaan, yksikkötestit jakaantuvat *kolmeen* luokkaan.

Ensimmäinen testien joukko varmistaa, että nuotti-instanssien muodostama nuottikuva on oletetun kaltainen.

Esim. jos syöte on "g2 d2", odotetaan että nuottiolion kuva on

```
  @@
-- | -----
  |
-- | -----@@----
  |       |
-- | ----- | ----
    =====
-----

----- (viivastopalan pituus voi vielä muuttua)
```

Vastaavasti testataan kattavasti eri aika-arvoilla, tauoilla sekä soinnulla.

Toisen testattavan kategorian muodostaa virheellisten syötteiden testaus. Testaus kohdistuu TiedostonLukija-luokkaan, jolle annetaan erilaisia virheellisiä syötteitä (esim.. virheellinen nuotin nimi) ja varmistetaan että syötteen lukeminen aiheuttaa poikkeuksen, joka näkyy käyttäjälle informaationa siitä kuinka virheen voi myös korjata (vrt. Käyttötapaus #2).

Seuraavaksi hieman hahmottelen yksikkötestauskoodia. Samoja testiehtoja voi ajaa joko hyväksyttävillä tai hylättävillä syötteillä. Molemmat ryhmät on hyvä olla jotta näkee onko itse testi toimiva, eikä vahingossa anna väärää totuusarvoa. Itse asiassa koodin viimeinen if ei vielä toimi, palaan siihen piakkoin, koska tuontapaisella koodilla pitää selvittää myös itse ohjelman kelvolliset syötteet.

```
"NuottiPiirturi" should "find non-valid note names (not investigating length)" in {
  val nuottejaVaarin = Buffer("t1-", "d3-", "f", "p", "g", "1c", "2f", "f2f2", "f2f", "f2f----", "hb22----")
  val nuottejaOikein = Buffer("c1-", "d1-", "f#2", "Gb1----", "Ab1--", "d#1---", "d1#----", "d1----#")

  var virheita = 0
  for (nuotti <- nuottejaVaarin) { // tähän voi vaihtaa nuottejaOikein-muuttujan
    println(nuotti.filter(_ != '-'))
    if(!"cdefgah".contains(nuotti.toLowerCase().head.toString()))
      virheita += 1
    else if(!(nuotti.tail.contains("1") || nuotti.tail.contains("2")))
      virheita += 1
    else if(nuotti.filter(_ != '-').size < 2)
      virheita += 1
    else if(nuotti.filter(_ != '-').size > 3)
      virheita += 1
    else if(! (nuotti.filter(_ != '-').size == 3 && ( nuotti.tail.contains("#") || nuotti.tail.contains("b") ) ) )
      virheita += 1
  }
  assert(virheita == nuottejaVaarin.size) // tähän voi vaihtaa nuottejaOikein ja odottaa virheita 0
}
```

Poiketen yleissuunnitelmasta - keksin tämän juuri tänään - olisi käyttäjäystävällistä hyväksyä nuotinnimet *c#1-*, *c#-1* ja *c1#-*, koska käyttäjä on lähes 100% varmuudella tarkoittanut juuri säveltä *cis1*. Tämän tyyppisiä nimiä löytyy yllä muuttujasta *nuottejaOikein*. Tätä asiaa pitää vielä pohtia ja testilla ajan kanssa, siis mitä kaikkea voi sallia niin että kuitenkin ohjelma osaa myös hylätä epäkelvot syötteet. Nytkin if-haarojen määrä ja kompleksisuus uhkaa karata käsistä, tai itse asiassa karkasi jo, koska testi ei toimi (vielä...)

Syötteitä voi lisätä puskuriin rajattoman määrän, aina kun keksii uuden "case":n. Näin ei tarvitse tehdä uutta testiä jokaiselle väärälle syönteelle.

Kolmas testien ryhmä muodostuu erilaisten raja-arvojen toiminnan varmistamisesta. Luodaan esim. *nuotteja* jotka ovat käsittelyalueen äärirajoilla, ja varmistetaan että varsinkin nuottien varret piirtyvät oikein.

7. Kirjallisuusviitteet ja linkit

Scala for the Impatient (Cay S. Horstmann)