**Suunnitteludokumentti**

**Ohjelmoinnin harjoitustyö / Kuvakokomuokkain**

Mikko Nylén <mikko.nylen@gmail.com>

Helsingin yliopisto, tietojenkäsittelytieteen laitos

**Ohjaaja:** Jesse Lankila

28.01.2009

# Luokkarakenne



Luokkarakenteesta nähdään heti, että itse kuvien koon muokkaamiseksi tarkoitetun koodin ja käyttöliittymän välillä on hyvin vähän yhteyksiä.

Käyttöliittymä luo ja rekisteröi itsensä kuuntelijaksi *BatchResizer*-luokalle sekä toteuttaa rajapinnan *BatchProgressListener*. *BatchResizer*-luokka puolestaan tiedottaa sille rekisteröityjä kuuntelijoita eräajon etenemisestä. Itse kuvan koon muokkauksen *BatchResizer* hoitaa sille annetulla *ImageResizer*-luokan toteutuksella (kuvakokomuokkain). Tallentaminen haluttuun kuvaformaattiin puolestaan onnistuu annetulla *ImageSaver-*luokan toteutuksella (tallennin).

Se, mitä kuvakokomuokkainta tai tallenninta käytetään, riippuu käyttäjän valinnoista käyttöliittymässä. Jos esimerkiksi käyttäjä haluaa tallentaa kuvat PNG-muotoon sekä käyttää laatikkoon mahduttavaa kuvakoon muokkaajaa, olisivat käytettävät toteutukset *PNGImageSaver* ja *BoxFittingImageResizer*.

Luokkarakenne on alusta alkaen suunniteltu olemaan helposti laajennettavissa. Tuki lisätallennusformaateille voidaan toteuttaa perimällä *ImageSaver*-luokka ja eksoottisempia tapoja muokata kuvan kokoa voidaan toteuttaa toteuttamalla *ImageResizer*-rajapinta. Tämän jälkeen ainoa tarvittava operaatio on lisätä käyttöliittymään tuki näille.

# Rajapintojen kuvaus

mnylen.imagebatchresizer.resizer-paketin rajapinta on kuvattuna liitteenä olevassa *ClassDiagram.mnylen.imagebatchresizer.resizer.png*-tiedostossa UML-luokkadiagrammina.

*BatchResizer*-luokan rajapinta on ehkäpä kaikista tärkein, sillä se on tarkoitettu käyttöliittymän käytettäväksi. Tämän vuoksi rajapinta onkin todella yksinkertaistettu: eräajoon voi lisätä kuvia *addFile(File)*-metodilla ja eräajon voi aloittaa *start()*-metodilla. Lisäksi eräajo on mahdollista lopettaa missä tahansa vaiheessa *stop()-*metodilla.

Toinen tärkeä rajapinta on *ImageResizer*, jota toteuttavat luokat hoitavat itse kuvakoon muokkaamisen. Tämä rajapinta sisältää vain yksinkertaisen *resize(BufferedImage)*-metodin. Oletuksena ohjelmassa on kolme *ImageResizer*-rajapintaa toteuttavaa luokkaa: *DefaultImageResizer*, *AspectKeepingImageResizer* sekä *BoxFittingImageResizer*. Nämä toteuttavat aiemmin määrittelydokumentissa kuvaillut tavat muuttaa kuvan kokoa. Jokaisella on omat metodinsa haluttujen asetusten muuttamiseen tarvittaessa.

Kuvien tallentaminen hoidetaan abstraktin *ImageSaver*-luokan aliluokilla, jotka toteuttavat *save(BufferedImage, File)* ja *getExtension()*  -metodit. *getExtension()*-metodi palauttaa tiedostopäätteen, joka on ominainen kyseisen tallentimen tallentamille tiedostoille. *save(BufferedImage, File)*-metodi puolestaan tallentaa kuvan. Se ottaa parametrikseen tallennettavan kuvan sekä tiedoston, jossa alkuperäinen kuva sijaitsee. *ImageSaver* sisältää lisäksi metodin *getDestinationFile(File)*, joka määrittää mihin tiedostoon kuva tallennetaan.

*BatchProgressListener* on rajapinta, jota eräajoprosessia kuuntelevat luokat toteuttavat. Tälle on määritelty kaikki tärkeät metodit, joilla voidaan tiedottaa kuuntelijoita tärkeistä eräajoon liittyvistä tapahtumista, kuten tiedoston vaihtumisesta, kuvan tallentamisesta, virheen tapahtumisesta tai eräajon tilan vaihtumisesta.

*BatchEvent*-luokka ja sen aliluokat *BatchSaveEvent* sekä *BatchErrorEvent* kuljettavat tapahtumiin liittyvää tietoa kuten *”mitä tiedostoa käsiteltiin, kun tapahtuma tapahtui”.* Esimerkiksi *BatchSaveEvent-*luokan ilmentymä kertoo, mitä tiedostoa käsiteltiin ja minne se tallennettiin.

# Toiminnan hahmottelu

## Eri tavat kuvan koon muuttamiseen

Kuten määrittelydokumentissa kerrottiin, käyttäjä voi ohjelmasta valita kolmesta tavasta muuttaa kuvan kokoa. Nämä olivat:

* *Yksinkertainen tapa*, jossa käyttäjä syöttää ohjelmaan haluamansa leveyden sekä korkeuden kuville. Tässä tavassa ohjelma yksinkertaisesti muuttaa kuvan koon halutuksi, eikä kiinnitä huomiota siihen, säilyykö kuvan alkuperäiset mittasuhteet koon muuttamisen yhteydessä.
* Mittasuhteet säilyttävä tapa, jossa käyttäjä määrittää haluamansa pituuden kuvan pisimmälle sivulle ja ohjelma laskee itse lyhyemmälle sivulle pituuden siten, että mittasuhteet säilyvät.
* Laatikkoon mahduttava tapa, jossa käyttäjä määrittää laatikon mitat, joiden sisälle alkuperäinen kuva mahdutetaan siten, että mahdollisimman suuri osa laatikon pinta-alasta tulee käytetyksi ja kuvan mittasuhteet säilyvät.

Alla on esitetty algoritmit näille pseudokoodina:

### Yksinkertainen tapa

simple\_resize(image, new\_width, new\_height)

resized\_image <- image

width[resized\_image] <- new\_width

height[resized\_image] <- new\_height

return resized\_image

### Mittasuhteet säilyttävä tapa

resize\_long\_side(image, long\_side\_length)

new\_width <- new\_height <- Nil

if width[image] > height[image]

then

new\_width <- long\_side\_length

new\_height <- round(new\_width \* height[image]) / width[height])

else

new\_height <- long\_side\_length

new\_width <- round(new\_height \* width[image]) / height[image])

return simple\_resize(image, new\_width, new\_height)

Algoritmi katsoo ensin, kumpi kuvan sivuista on pisin ja asettaa tämän jälkeen kyseisen sivun pituudeksi halutun pisimmän sivun pituuden. Tämän jälkeen algoritmi laskee lyhyemmälle sivulle pituuden kaavalla: , jossa *l* on haluttu pisimmän sivun pituus, *x* on kuvan lyhyimmän sivun pituus ja *y* kuvan pisimmän sivun pituus.

### Laatikkoon mahduttava tapa

resize\_fit\_to\_box(image, box\_width, box\_height)

long\_side\_length <- Nil

if box\_width >= box\_height

then long\_side\_length <- box\_width

else long\_side\_length <- box\_height

resized\_image <- resize\_long\_side(image, long\_side\_length)

box\_image <- create\_image(box\_width, box\_height, color(‘white’))

paste\_to\_center(box\_image, resized\_image)

return box\_image

Tämä algoritmi puolestaan käyttää hyväkseen aiempaa kuvan mittasuhteet säilyttävää tapaa ja valitsee halutuksi pisimmän sivun pituudeksi laatikon pisimmän sivun pituuden. Muutettu kuva tallennetaan muuttujaan *resized\_image*, jonka jälkeen se liitetään keskelle valkoisella taustalla varustettua *box\_image*-kuvaa, jonka mitat ovat annetun laatikon mitat.

## Kuvakoon muokkausprosessi

Kun eräajo aloitetaan *start()*-metodilla, luodaan sille erillinen säie, jotta raskaat operaatiot eivät häiritsisi käyttöliittymän toimintaa. Tämän jälkeen prosessin tila muuttuu *Idle*-tilasta *Processing*-tilaan.

Nyt käydään läpi jokainen *addFile(File)*-metodilla lisätty kuva. Jokaiselle tiedostolle tehdään seuraava:

1. Tiedotetaan kuuntelijoita käsiteltävän kuvan vaihtumisesta *fileChanged()*-metodin kutsumisella.
2. Kuva ladataan tiedostosta.
   1. Jos epäonnistui, tiedotetaan kuuntelijoita virhetilanteesta *errorOccured()*-metodin kutsumisella ja jatketaan seuraavasta kuvasta (palataan kohtaan 1).
3. Muokataan kuvan kokoa kuvakokomuokkaimella.
4. Tallennetaan muokattu kuva tiedostoon tallentemille.
   1. Jos epäonnistui, tiedotetaan kuuntelijoita virhetilanteesta *errorOccured()*-metodin kutsumisella ja jatketaan seuraavasta kuvasta (palataan kohtaan 1).
5. Tiedotetaan kuuntelijoita kuvan tallentamisesta *fileSaved()*-metodin kutsumisella.

Lopulta prosessin tila muutetaan *Processing*-tilasta *Finished*-tilaan.

Käyttäjä voi missä tahansa välissä prosessia valita päättää kuvien muokkauksen. Tällöin käyttöliittymästä kutsutaan *BatchResizer*-luokan *stop()*-metodia, mikä aiheuttaa kuvien koon muokkauksen loppuvan heti, kun *stop()-*metodin kutsuhetkellä käsiteltävä kuva on muokattu loppuun. Myös tällöin prosessin tila muutetaan *Finished*iksi.

# Liitteet

Suunnitteludokumentin liitteeksi olen liittänyt seuraavat tiedostot:

* *ClassStructure.png* – luokkarakenne yksinkertaistettuna ilman rajapintakuvauksia
* ClassDiagram\_mnylen.imagebatchresizer.resizer.png – luokkien rajapintojen kuvaus.