5. Kuvien käsittely Javalla[1]

Perusluokat

Syöttö ja tulostus

Värimallit

---

[1]Huom. Tämän luvun asioita ei vaadita tentissä

Javan kuvankäsittelyn kehitys

- Java 1.0/1.1:
  - Suppea, kankea
- Java2D API:
  - Monipuolisempi
  - Kuvankäsittely, grafiikka, renderöinti (’pinnoitus’, ’muodostus’), teksti, ym.
- Java Advanced Imaging (JAI)
  - Korkean tason välineitä, esim. suodattimet, muunnokset

Kuvan lataus muistiin

- Verkosta (URL-osoite) tai paikallisesta tiedostosta.
- Lataustavat:
  - *asynkroninen* (Java 1.0/1.1)
  - *synkroninen*
- Formaatteina GIF, JPEG tai PNG
- Latausmetodi esim. *getImage*(...)
  - Asynkroninen, ts. välitön return, ja kuvan lataus eri säikeessä.

Java 1.0/1.1: Kuvan lataus

- Appletti:
  - *getImage*(*getCodeBase*(), ”*kuva.jpg*")
  - Appletin URL-osoitteesta; tuloksena Image-objekti
- Sovellus:
  - *Toolkit*-luokan *getImage*(...) –metodi; asynkroninen lataus
  - awt-komponenttien (kuten *Applet*) toteuttama *ImageObserver*-rajapinta tarjoaa lataajalle metodin *imageUpdate*(), jolla ladattu kuva saadaan käyttöön.
  - Latauksen seuranta: *MediaTracker*,
    Kuvan otto seurantaan: *addImage*(kuva, no),
    Kuvan lataustesti: *waitForID*(no)

Kuvan lataus- ja näyttöappletti

```
import java.applet.Applet;
import java.awt.image.*;
import java.awt.*;
public class KuvaApplet extends Applet
{
  Image kuva;

  public void init()
  { kuva = getImage(getCodeBase(),”kuva.jpg”); }

  public void paint (Graphics g)
  { g.drawImage (kuva, 0, 0, this); } // this: ImageObserver
}
```

Appletin kutsu 1: Ennalta kiinnitetty kuva

```
<HTML>
   <HEAD> <TITLE>KuvanHaku</TITLE> </HEAD>
   <BODY>
     <H1>Kuvanhakuappletti</H1>
       <APPLET CODE="KuvaApplet.class"
                       WIDTH=616 HEIGHT=396>
         <PARAM NAME=src VALUE="Joki.jpg">
       </APPLET>
   </BODY>
</HTML>
```

Appletin kutsu 2: Käyttäjä valitsee kuvan

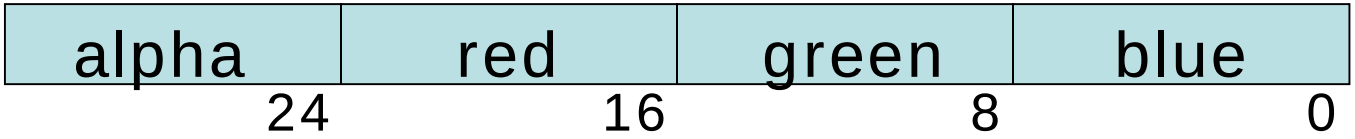
```
<HTML>
   <HEAD> <TITLE>KuvanHaku</TITLE> </HEAD>
   <BODY>
    <H1>Kuvanhakuappletti</H1>
      <SCRIPT  language=javascript  type=text/javascript>
        <!-- var tdsto = prompt("Anna kuvatiedoston nimi", "");
          document.write("<APPLET CODE='KuvaApplet.class'
                                        WIDTH=800 HEIGHT=500>");
          document.write("<PARAM NAME=src
                                        VALUE="'+tdsto+"'>");
          document.write("</APPLET>");
        // -->
      </SCRIPT>
   </BODY>
</HTML>
```

Kuvan lataussovellus

```
import java.awt.*;
public class KuvaTesti {
    public static Image lueKuva(String tdsto) {
      Image kuva = Toolkit.getDefaultToolkit().getImage(tdsto);
      Mediatracker tr = new MediaTracker
          (new Component() { }); // Vaatii Component-param.
      tr.addImage(kuva, 0);
      try { tr.waitForID(0); }
          catch (InterruptedException e) { }
      return kuva;
 }}
```

AWT:n värimallit: Luokka ColorModel

- Aliluokkia:
  - *DirectColorModel* (-> *PackedColorModel*): Normaali RGB-esitys + alfa-kanava (läpinäkyvyys); pakkaus esim. int-muuttujaan.



  - *IndexColorModel*: Indeksoitu värimalli, pikselit viittauksia väritauluun (= paletti), josta löytyvät todelliset RGB (& α-arvot); sopii mm. harmaasävykuville.
  - *ComponentColorModel*: yleinen värimalli

AWT:n Color-luokka

- Värin esittäminen (oletus RGB) ja muunnokset
- Metodeja:
  getRed(), *getGreen*(), *getBlue*(), *getAlpha*()
- Muunnos RGB $\leftrightarrow$ HSB
  (HSB = *Hue-Saturation-Brightness* $\approx$ HSI):
  - float[ ] hsb=Color.RGBtoHSB(r, g, b, null)
    Tulos sis. kolme liukulukua 0.0...1.0.
  - int rgb = Color.HSBtoRGB(h, s, b);

Tuottaja-kuluttaja-paradigma

- Tuottaja tuottaa pikseleitä kuluttajalle. Rajapinnat:
  - *ImageProducer*
  - *ImageConsumer*
- Kuluttaja voi esim. näyttää kuvan awt-komponenttien avulla.
- Välivaiheena voi olla esim. suodatus
  - *ImageFilter* suodattaa
  - *FilteredImageSource* tuottaa suodatetun tuloksen

Tuottajaesimerkki: harmaasävykuvan luonti

```
byte[ ] pikselit = ...;  // Kuvadata (8 bittiä/pikseli)
byte[ ] harmaa = new byte[256];
for (int i=0; i<256; i++)
    harmaa[i] = (byte) i;
ColorModel harmaaSavyt = new IndexColorModel (8, 256,
    harmaa, harmaa, harmaa);
ImageProducer tuottaja =
    new MemoryImageSource(200, 150,
        harmaaSavyt, pikselit, 0, 200);
Image kuva = Toolkit.getDefaultToolkit().
    createImage(tuottaja);
```

Yleisiä suodattimia

- *ImageFilter* (suodattimien yliluokka; null-suodatin)
- *RGBImageFilter*: pikseliarvojen muutokset (metodi *filterRGB*)
- *ReplicateScaleFilter*: skaalaus (rivien/sarakkeiden monistus/poisto)
- *AreaAveragingScaleFilter*: skaalaus & pehmennys (’anti-aliasointi’)
- *CropImageFilter*: kuvan suorakulmiorajaus

Pikselien ottaminen käsittelyyn (kuluttaja)

- *getImage*() tuottaa kuvan *asynkronisesti*, joten ei tiedetä, milloin pikselit ovat käytettävissä.
- *PixelGrabber*-luokan metodi *grabPixels*() odottaa, kunnes pikselit ladattu.
- Viittaus pikselitauluun saadaan grabberin metodilla *getPixels*().
- **Huom!** *PixelGrabber* toteuttaa *ImageConsumer*-rajapinnan

Pikselien noutoesimerkki

```
Image kuva = Toolkit.getDefaultToolkit().
    getImage(tdsto);
try {
    PixelGrabber gr =
        new PixelGrabber(kuva, 0, 0, -1, -1, false);
    if (gr.grabPixels()) {
        int w = gr.getWidth();
        int h = gr.getHeight();
        int data[] = (int[]) gr.getPixels();
    }
} catch (InterruptedException e) { ... }
```

Värikomponenttien poiminta

Pikselin *data*[*i*] RGB-arvojen poiminta:

```
int red = (data[i] & 0xff0000) >> 16;
int green = (data[i] & 0xff00) >> 8;
int blue = data[i] & 0xff;
```

Yleisempi toteutus:

```
PixelGrabber gr = new ... ;
ColorModel m = gr.getColorModel();
int red = m.getRed(data[i]);
int green = m.getGreen(data[i]);
int blue = m.getBlue(data[i]);
```

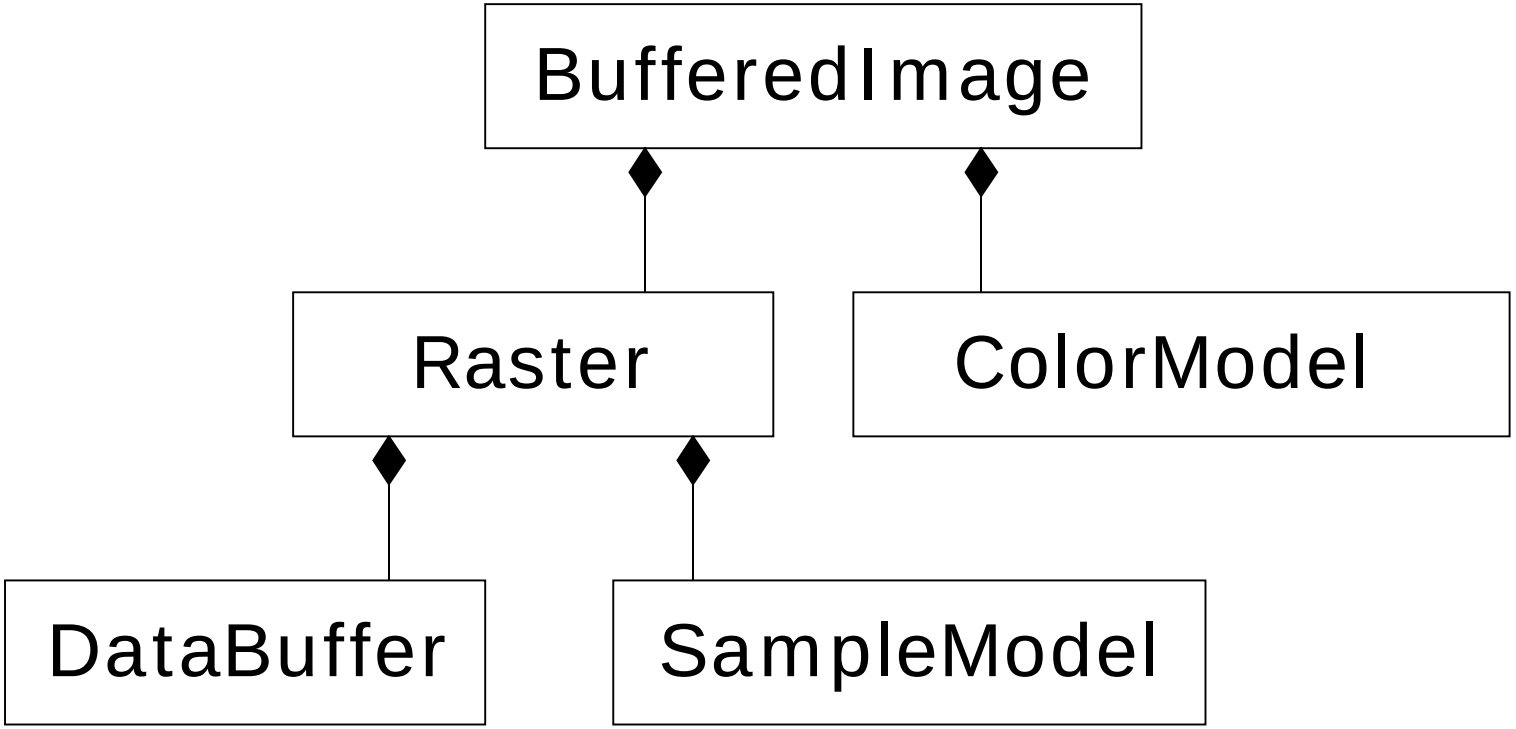
Java2D API:n pakkaukset

- *java.awt* (*Abstract Windowing Toolkit*)
- *java.awt.image* (laajennettu)
- *java.awt.color*
- *java.awt.font*
- *java.awt.geom*
- *java.awt.print*
- *java.awt.renderable*
- *javax.imageio* (kuvan luku & dekoodaus, koodaus & kirjoitus)

BufferedImage-luokka (Java2D)

- Imagen aliluokka; ero: pikselit heti käytettävissä (’immediate-moodi’).
- Komponentit:
  - Värimalli (*ColorModel*)
  - Mallin mukaiset pikselit (*Raster*)
- Värimallit:
  - *DirectColorModel* (RGB + alpha pakattu nippuun)
  - *IndexColorModel* (indeksit palettiin)
  - *ComponentColorModel* (värikomponentit eri tietoalkioissa; mielivaltainen ColorSpace)

BufferedImage-luokan rakenne



BufferedImage-luokka (jatk.)

- Rasterin komponentit:
  - *DataBuffer* (pikselitaulukko)
  - *SampleModel* (alkioiden eli värikomponenttien muunto pikseleiksi)
- Luontiparametrit:
  - Kuvan dimensiot
  - Kuvatyyppi (14 erilaista; nimettyjä int-vakioita),esim.
    - TYPE_BYTE_GRAY
    - TYPE_INT_ARGB
    - TYPE_BYTE_BINARY
    - TYPE_BYTE_INDEX

BufferedImage-luokka (jatk.)

- ’Getterit’:
  - *getWidth*(), *getHeigt*(), *getType*(), *getColorModel*(), *getRGB*(), *getData*()
- ’Setterit’:
  - *setRGB*(), *setData*()

- Mahdollinen tehottomuus:
  - Jos käytössä *ComponentColorModel*, niin värikomponenttien manipulointi on työlästä.
  - Harmaasävykuvan pikselit 4 komponentilla.

Raster-komponentin käyttö

- Käsittelemällä Rasteria suoraan vältetään mahdollinen tehottomuus värimallin tulkinnassa.
- Raster-luokka read-only; metodeja: *getPixel*(), *getPixels*(), *getSample*(), *getSamples*()
  [sample = pikselin värikomponentti]
- Aliluokka: WritableRaster;
  metodeja: *setPixel*(), *setPixels*(), *setSample*(), *setSamples*()

Esim. harmaasävykuvan vaalennus tai tummennus (sävy + delta)

```
BufferedImage image = ... // Jostain
WritableRaster wr = image.getRaster();
int uusi;
for (int x=0; x<image.getWidth(); x++)
   for (int y=0; y<image.getHeight(); y++) {
        // Harmaasävy värikanavalta 0
        uusi = wr.getSample(x, y, 0) + delta;
        if (uusi > 255) uusi = 255;
        wr.setSample(x, y, 0, uusi)
   }
```

Kuvien näyttö Javassa

- Periaatteessa yksinkertaista
- Graphics- tai Graphics2D-komponenteilla; instanssit edustavat pintaa, jolla kuva (rasterikuva, piirros, teksti) voidaan näyttää
- Näyttö yleensä kuvaruudulla
- AWT-komponentit alustariippuvia, Swing-komponentit eivät.

Kuvien näyttö AWT:lla

- Määritellään komponentin *paint*()-metodi (uudelleen) tulostamaan haluttu kuva.
- Komponentteja *Applet*, *Frame*, *Canvas*, ...
- *paint*()-metodin kutsu tapahtuu automaattisesti tarvittaessa (kun ikkunan koko tai näkyvyys muuttuu), tai komentamalla *repaint*().
- Piirtometodin perusmuoto (ks. aik.):

  public boolean drawImage (Image im,
  int x, **int** y, ImageObserver obs)

Esim. Luokka, jolla voi piirtää Canvas-alustalle

```
import java.awt.*;
public class KuvaCanvas extends Canvas
{
   BufferedImage kuva;
   public KuvaCanvas(BufferedImage i)
    { kuva = i; }
   public void paint(Graphics g)
    { g.drawImage(kuva, 0, 0, this); }
}
```

Esimerkki piirtosovelluksesta

```
import java.awt.*;
import java.awt.image.*;
import java.lang.*;

public class Tulosta
// Sovellus näyttää kuvatiedoston,
// joka annetaan komentoriviparametrina.
// Perus-awt-toteutus.
{

    public static Image lueKuva(String tdsto) {
      ...
    } // End of lueKuva



    public static class KuvaCanvas extends
      Canvas {
        …
    } // End class KuvaCanvas

    public static void main(String[] argv) {
    // Pääohjelma lukee kuvan, luo kehyksen
    // ja laittaa kuvan siihen näkyviin Canvas-
    // komponenttina.
       if (argv.length > 0) {
          Image i = lueKuva(argv[0]);
          KuvaCanvas kc = new KuvaCanvas(i);
          Frame kehys = new Frame(argv[0]);
          int leveys = i.getWidth(kc);
          int korkeus = i.getHeight(kc);
          kc.setSize(leveys, korkeus);
          kehys.setLayout(new BorderLayout());
          kehys.setBackground(Color.gray);
          kehys.add(
                kc, BorderLayout.CENTER);
          kehys.setVisible(true);
       }
    } // End of main
} // End class Tulosta
```

drawImage()-variaatioita

- Skaalaus tiettyyn kokoon (w, h):

  public boolean drawImage(
  Image im, **int** x, **int** y,
  int w, **int** h, ImageObserver obs)

- Kuvankäsittelyoperaatio ennen näyttöä (vain Graphics2D:ssa):

  public void drawImage(BufferedImage im,
  BufferedImageOp op, **int** x, **int** y)

Esim. Yleistetty KuvaCanvas

```
import java.awt.*;
import java.awt.image.*;
public class KuvaCanvas extends Canvas
{   BufferedImage kuva;
    BufferedImageOp oper;
    public KuvaCanvas(BufferedImage i,
          BufferedImageOp op)
     { kuva = i; oper = op; }
    public void paint(Graphics g)
     { Graphics2D g2 = (Graphics2D) g;
          g2.drawImage(image, oper, 0, 0); }
}
```

Kuvan näyttö Swing-komponenteilla

- Pakkaus *javax.swing.**
- Komponentit alustariippumattomia
- Ei kannata sekoittaa keskenään AWT:n ja Swingin visuaalisia komponentteja.
- Esim. *JLabel*-komponentilla voidaan näyttää kuva:

```
BufferedImage im = ...; // Saadaan jostain
JLabel kuva =
    new JLabel (new ImageIcon(im));
```

Esimerkki: Display-luokka

```
import java.awt.*;
import java.awt.image.*;
import javax.swing.*;

public class Display extends JFrame {
    public Display(String tdsto) {
        super(tdsto);  // Frame-otsikko
        BufferedImage im = ... ;  // Kuvan nouto jotenkin
        JLabel kuva = new JLabel(new ImageIcon(im));
        getContentPane().add(kuva);
        addWindowListener(...);  // Varaudu sulkemiseen
    }
}
```

Java ja kuvaformaatit: syöttö & tulostus tiedostoon

- *javax.imageio* –pakkaus:
  - Koodekit formaateille: JPEG, PNG, BMP, WBMP, GIF
  - Formaattien välinen transkoodaus

- javax.imageio.ImageIO -luokka
  - Staattiset metodit joilla löydetään *ImageReader*it ja *ImageWriter*it
  - Yksikertaiset koodaus- ja dekoodaustoiminnot

Esimerkki: Kuva negatiivikuvaksi

```
import java.io.*;
import java.awt.*;
import java.awt.image.*;
import java.awt.event.*;
import javax.swing.*;
import javax.imageio.*;

public class ImageIOtesti extends JFrame {
// JPEG-testiluokka: kuvan lukeminen, muunto
// negatiiviksi, näyttö ja kirjoitus tiedostoon.
BufferedImage im;  // Kuvaobjekti
//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
public ImageIOtesti(String tdsto) {
// Lukee kuvan tdsto:sta ja asettaa sen JLabel-
// komponenttiin, joka asetetaan JFrameen.
  super(tdsto);  // Frame-otsikko
  try {
    im = ImageIO.read(new File(tdsto));
    JLabel kuva = new JLabel(new ImageIcon(im));
    getContentPane().add(kuva);
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent event)
      { System.exit(0); }
    });
  } catch (Exception e) { System.out.println(e); }
} // End of constructor
//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
public void negatiiviksi() {
// Jokainen värikomponentti vähennetään 255stä
  WritableRaster wr = im.getRaster();
  for (int x=0; x<im.getWidth(); x++)
    for (int y=0; y<im.getHeight(); y++)
      for (int rgb=0; rgb<3; rgb++) {
          int uusi = 255-wr.getSample(x, y, rgb);
          wr.setSample(x, y, rgb, uusi);
      }
} // End of negatiiviksi
//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
public void kirjoita() {
// Kirjoitetaan  nykyinen kuva tiedostoon
// "Muutettu.jpg".
  try {
       File newFile = new File("Muutettu.jpg");
       ImageIO.write(im, "JPG", newFile);
  } catch (Exception e) {
    System.out.println(e);
  }
} //End of kirjoita
```

Esimerkin pääohjelma

```
public static void main(String[] args) {
// Pääohjelma luo kehyksen, jolla kuva näytetään,
// kutsuu negatiivimuunnosta ja näyttää kuvan.
// Lisäksi kirjoitetaan muutettu kuva tiedostoon..
  try {
    ImageIOtesti fr = new ImageIOtesti(args[0]);
    fr.pack();
    fr.setVisible(true);
    Thread.sleep(2000);  // Alkup. kuva 2 sek
    fr.negatiiviksi();
    fr.repaint();        // Uusi kuva näyttöön
    fr.kirjoita();       // Tiedostoksi
  } catch (Exception e) {
    System.out.println(e);
  }
}
} // end class ImageIOtesti
```