Images

- Lire une image
- Images d'icones
- Afficher une image
- Création d'images tampon
- Traitement d'images

Lire une image

- Lecture simple dans un fichier

```
String nom = "bleu.gif";
Image image;
image Toolkit.getDefaultToolkit().getImage(nom);
```

- Lecture sur Internet

```
URL u = new URL("http://www.quelquepart.com/image.jpg");
Image image = Toolkit.getDefaultToolkit().getImage(u);
```

- Dessin de l'image, à l'aide d'un contexte graphique, dans un composant

```
graphics.drawImage(image, 0, 0, null);
```

Attendre l'image

- La requête **getImage()** est enregistrée, mais son exécution est retardée.
- Le chargement de l'image a lieu au premier appel de **drawImage**(), mais
  - le chargement est pris en main par un nouveau thread, en asynchrone
  - l'exécution du programme continue sans attendre la fin du chargement.
- L'affichage peut donc être progressif.
- Pour attendre la fin du chargement, on utilise un "pisteur" de la classe **MediaTracker**.

Pister le chargement

- Les images à pister sont ajoutées au pisteur, chacune avec un numéro d'identification

```
MediaTracker tracker = new MediaTracker(this);
Image image = Toolkit.getDefaultToolkit().getImage(nom);
tracker.addImage(image,0);
```

l'argument est le composant dans lequel l'image sera affichée

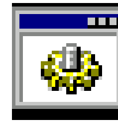
- On attend la fin du chargement d'une image par

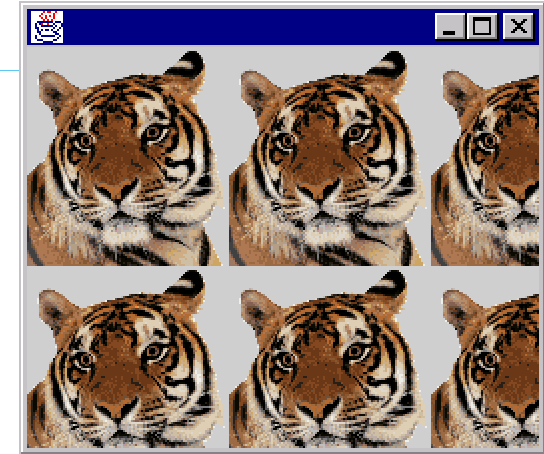
```
try { tracker.waitForID(0); }
catch (InterruptException exc) {}
```

ou de toutes les images pistées plus simplement par

```
try { tracker.waitForAll(); }
catch (InterruptException exc) {}
```

Exemple





```
class ImagePanel extends JPanel {
   private Image image;

   public ImagePanel() {
      image = Toolkit.getDefaultToolkit().getImage("tiger.gif");
      MediaTracker tracker = new MediaTracker(this);
      tracker.addImage(image, 0);
      try { tracker.waitForAll(); }
      catch (InterruptedException e) {}
   }
   public void paintComponent(Graphics g) {
      super.paintComponent(g);
      int width = getSize().width;
      int height = getSize().height;
      int imageWidth = image.getWidth(this);
      int imageHeight = image.getHeight(this);

      g.drawImage(image, 0, 0, this);
      for (int w = 0; w < width; w += imageWidth)
         for (int h = 0; h < height; h += imageHeight)
            if (w + h > 0)
               g.copyArea(0, 0, imageWidth, imageHeight, w, h);
   }
}
```

Icones

- La classe **javax.swing.ImageIcon** est une implémentation de l'interface **javax.swing.Icon.**
- Les constructeurs

```
public ImageIcon(String nomFichier)
public ImageIcon(URL url)
public ImageIcon(byte[] donneesImage)
```

attendent que l'image soit chargée avant de retourner.

- La méthode **getImage()** retourne l'image représentée par l'objet.
- Ainsi, le chargement d'une image s'écrit simplement

```
Image image = new ImageIcon("tiger.gif").getImage();
```

Afficher une image

- Il y a 6 versions de **drawImage**

```
public boolean drawImage(Image img, int x, int y, ImageObserver observer)
```

  - affiche l'image à l'endroit indiqué

```
public boolean drawImage(Image img, int x, int y, Color bgcolor,
   ImageObserver observer)
```

  - les parties transparentes sont rendues dans la couleur de fond

```
public boolean drawImage(Image img, int x, int y, int width, int height,
   ImageObserver observer)
```

  - l'image est déformée pour remplir le rectangle donné. Variante avec couleur de fond

```
public boolean drawImage(Image img, int dx1, int dy1, int dx2, dy2, int sx1, int
   sy1, int sx2, int sy2, ImageObserver observer)
```

  - la partie de l'image délimitée par le rectangle donné par les points d1 et d2 est affichée dans le rectangle donné par les points s1 et s2. Une déformation peut avoir lieu. Variante avec couleur de fond

Manipulation d'images

- Un objet de la classe `Image` n'est pas modifiable.
- La classe `java.awt.image.BufferedImage` permet cet accès. C'est une classe dérivée de la classe `Image`.
- C'est une situation semblable à `String` et `StringBuffer`

- Une image tampon est un tableau rectangulaire de pixels.
  - A chaque pixel est associé la couleur d'un point, dans une parmi plusieurs formes possibles appelées ***valeurs d'échantillonage***.
  - Ces valeurs sont interprétées selon le ***modèle de couleur*** de l'image.

Créer une image tampon

- Création d'une image vide

```
BufferedImage(int width, int height, int typeImage)
```

- Les types d'images indiquent comment les couleurs des pixels sont codées.

`TYPE_3BYTE_BGR`	bleu, vert, rouge, sur 8 bits chacun
`TYPE_4BYTE_ABGR`	alpha, bleu, vert, rouge, sur 8 bits chacun
`TYPE_4BYTE_ABGR_PRE`	alpha, bleu, vert, rouge, sur 8 bits chacun, les couleurs pondérées
`TYPE_BYTE_BINARY`	1 bit par pixel, groupés en octets
`TYPE_BYTE_INDEXED`	1 octet par pixel, indice dans une table de couleurs
`TYPE_BYTE_GRAY`	1 octet par pixel, niveau de gris
`TYPE_USHORT_555_RGB`	rouge, vert, bleu, sur 5 bits, codés dans un short
`TYPE_USHORT_565_RGB`	rouge sur 5, vert sur 6, bleu sur 5 bits
`TYPE_USHORT_GRAY`	niveau de gris sur 16 bits
`TYPE_INT_RGB`	rouge, vert, bleu sur 8 bits chacun, dans un int
`TYPE_INT_BGR`	bleu, vert, rouge (Solaris)
`TYPE_INT_ARGB`	alpha, rouge, vert, bleu sur 8 bits, dans un int
`TYPE_INT_ARGB_PRE`	les couleurs déjà pondérées par alpha

Création d'une image tampon

- Exemple standard : création d'une image (vide)

```
BufferedImage image = new BufferedImage(500, 400, BufferedImage.TYPE_INT_RGB);
```

- On accède aux pixels par une objet **`WritableRaster`**, par

```
WritableRaster raster = image.getRaster();
```

- On fixe la valeur d'un pixel par **`setPixel()`**, mais en donnant la couleur dans le ***type*** de l'image.
  - Dans le type **`TYPE_INT_ARGB`**, on fournit un tableau de 4 entiers entre 0 et 255, pour les composantes alpha, rouge, vert, bleu:

```
int[] bleu = { 0, 0, 0, 255};
raster.setPixel(i, j , bleu);
```

Création d'une image tampon

- Le type de l'image est encapsulé dans le modèle de couleur
- La méthode **`getDataElements()`** du *modèle* fait la conversion, et la méthode **`setDataElements()`** du *raster* équivaut à **`setPixel()`**

  - Obtention du modèle

```
ColorModel model = image.getColorModel();
```

  - Transformation de couleur en valeurs d'échantillonage

```
Color couleur = new Color(0.0f, 0.0f, 1.0f);
int argb = couleur.getRGB();
Object donnee = model.getDataElements(argb, null);
```

  - Affectation des valeurs d'échantillonage

```
raster.setDataElements(i, j, donnee);
```

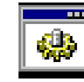
Création d'une image tampon

- Création à partir d'une image de type `Image`

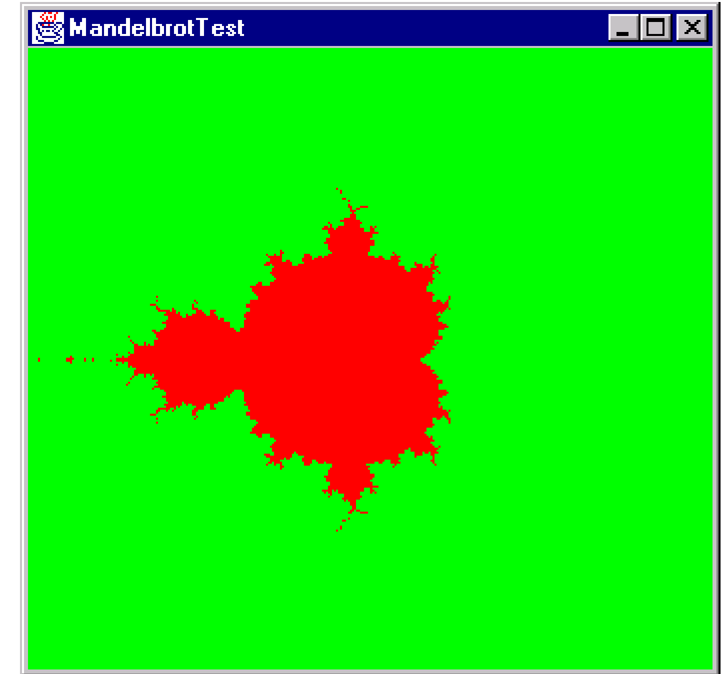
```
static BufferedImage makeBufferedImage(Image image, int imageType) {
  BufferedImage bufferedImage;
  bufferedImage = new BufferedImage(
      image.getWidth(null),image.getHeight(null), imageType);
  Graphics2D g2 = bufferedImage.createGraphics();
  g2.drawImage(image, null, null);
  return bufferedImage;
}
```

`createGraphics` est comme `getGraphics`

Exemple Mandelbrot



- Le panneau contenant l'image




```
class MandelbrotPanel extends JPanel {
   public void paintComponent(Graphics g) {
      super.paintComponent(g);
      setBackground(Color.green);
      BufferedImage image = new BufferedImage(getWidth(),
         getHeight(), BufferedImage.TYPE_INT_ARGB);
      generer(image);
      g.drawImage(image, 0, 0, null);
   }

   public void generer(BufferedImage image) {...}
   ...
}
```

Exemple Mandelbrot (suite)

- La fonction generer détermine les points de la fractale et les ajoute à l'image.

```
class MandelbrotPanel {
   public void generer(BufferedImage image) {
      int width = image.getWidth();
      int height = image.getHeight();
      WritableRaster raster = image.getRaster();
      ColorModel model = image.getColorModel();

      Color fractalColor = Color.red;
      int argb = fractalColor.getRGB();
      Object colorData = model.getDataElements(argb, null);

      for (int i = 0; i < width; i++)
         for (int j = 0; j < height; j++) {
            double a = XMIN + i * (XMAX - XMIN) / width;
            double b = YMIN + j * (YMAX - YMIN) / height;
            if (converge(a, b))
               raster.setDataElements(i, j, colorData);
         }
   }
   ...
}
```

Exemple Mandelbrot (fin)

- L'aspect algorithmique est couvert par la fonction `converge()`

```
class MandelbrotPanel {
   ...
   private boolean converge(double a, double b) {
      double xnew, x = 0.0;
      double ynew, y = 0.0;
      int iters = 0;
      for (int iters = 0; iters < MAX_ITERATIONS; iters++) {
         xnew = x * x - y * y + a;
         ynew = 2 * x * y + b;
         x = xnew;
         y = ynew;
         if (x > 2 || y > 2) return false;
      }
      return true;
   }

   private static final double XMIN = -2;
   private static final double XMAX = 2;
   private static final double YMIN = -2;
   private static final double YMAX = 2;
   private static final int MAX_ITERATIONS = 16;
}
```

Structure d'une BufferedImage

- Une **BufferedImage** est composée
  - d'un **ColorModel** qui définit la façon d'interpréter les couleurs
  - d'un **WritableRaster**, donc d'un raster autorisé en écriture.
- Un **Raster** est composé
  - d'un **DataBuffer** contenant les données brutes, dans un tableau
  - d'un **SampleModel** (modèle d'échantillonage) qui interprète les données brutes.
- On obtient le modèle et le raster par les méthodes **get**.

```
WritableRaster raster = image.getRaster();

ColorModel model = image.getColorModel();
```

Traitement d'images

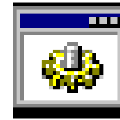
- Java propose des opérations de traitement d'images (filtres).
- L'interface est **BufferedImageOp**. Appliquer une opération est le *filtrage*.

```
BufferedImageOp operation = ...;
BufferedImage imageFiltree = operation.filter(image);
```

- Il existe cinq classes qui implémentent **BufferedImageOp**:

  AffineTransformOp
  RescaleOp
  LookUpOp
  ColorConvertOp
  ConvolveOp

Convolutions



- Les plus spectaculaires sont le lissage (*blur*) et la détection de contour
- Une convolution calcule la valeur pour chaque pixel en pondérant les couleurs des 9 pixels de son entourage.
- Pour le flou, les 9 pixels ont même poids.
- Pour la détection de contour, les voisins ont poids nul ou négatif.
- Les valeurs sont données dans un vecteur de flottant appelé *noyau*.

Noyaux des convolutions

- Pour le lissage :

```
float[] blur =
   {1f/9f, 1f/9f, 1f/9f,
    1f/9f, 1f/9f, 1f/9f,
    1f/9f, 1f/9f, 1f/9f};

Kernel kernel = new Kernel(3, 3, blur);
ConvolveOp op = new ConvolveOp(kernel);
BufferedImage imageFiltree = op.filter(image);
```

- Pour la détection des contours:

```
float[] contour =
   { 0f, -1f, 0f,
     -1f, 4f, -1f,
     0f, -1f, 0f};
```

- Pour l'accentuation des couleurs:

```
float[] sharpen =
   { 0f, -1f, 0f,
     -1f, 5f, -1f,
     0f, -1f, 0f};
```

fabriquer les convolutions

- On les fabrique dans une classe de fabrication:

```
class ConvolutionFabrique {
  public static ConvolveOp createLisseur() {
    float[] blur = { 1f/9f, 1f/9f, 1f/9f,
                     1f/9f, 1f/9f, 1f/9f,
                     1f/9f, 1f/9f, 1f/9f };
    return new ConvolveOp(new Kernel(3, 3, blur), ConvolveOp.EDGE_NO_OP, null);
  }
  public static ConvolveOp createAiguiseur() {
    float[] sharp = { 0f, -1f, 0f,
                     -1f, 5f, -1f,
                      0f, -1f, 0f };
    return new ConvolveOp(new Kernel(3, 3, sharp));
}
```

constructeurs d'une convolution

- Une convolution est définie par un ***noyau***, et
  - un comportement sur les bords
  - éventuellement des indications sur le rendu (**`RenderingHints`**)
- Deux constructeurs :

```
ConvolveOp(Kernel kernel)
ConvolveOp(Kernel kernel,
      int edgeCondition, RenderingHints hints)
```

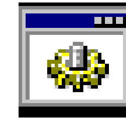
- les conditions au bords sont :

`EDGE_NO_OP` les pixels aux bords sont copiés sans modification
`EDGE_ZERO_FILL` les pixels aux bords sont traités comme s'ils étaient entourés de zéros (défaut).

- Exemples:

```
op = ConvolveOp(new Kernel(3, 3, blur),
                ConvolveOp.EDGE_NO_OP, null);
op = ConvolveOp(new Kernel(3, 3, sharp));
```

rescale



- Les opérations de “rescaling” modifient les intensités des composantes RGB par deux paramètres :
  - un facteur multiplication **`m`**
  - un décalage ***d***
- La nouvelle intensité est **`i' = i * m + d`**
- Ainsi,
  - si **`m`** < 1, l'image est assombrie
  - si **`m`** > 1, l'image est plus brillante
  - **`d`** est compris entre 0 et 256 et ajoute un éclairement supplémentaire
- Exemples:

```
op = new RescaleOp(.5f, 0, null)        plus sombre
op = new RescaleOp(.5f, 64, null)       plus sombre avec décalage
op = new RescaleOp(1.2f, 0, null)       plus brillant
op = new RescaleOp(1.5f, 0, null)       encore plus brillant
```

Exemple

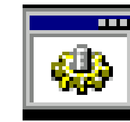
- plus sombre

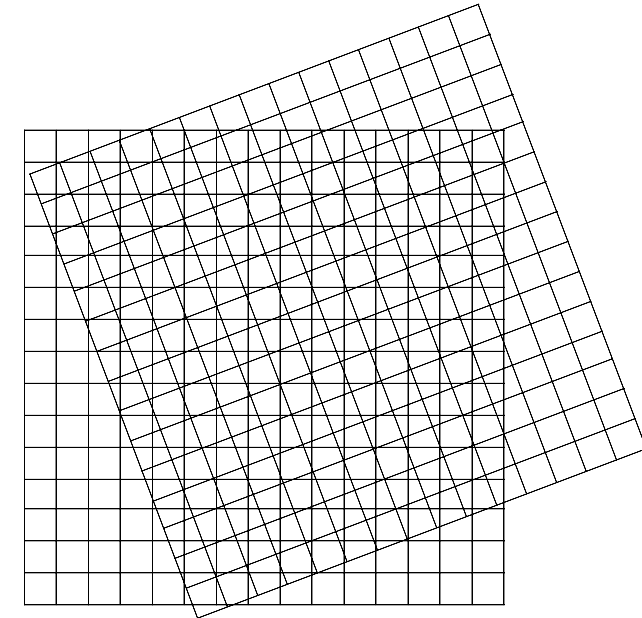
```
RescaleOp(.5f, 0, null)
```

- plus brillant

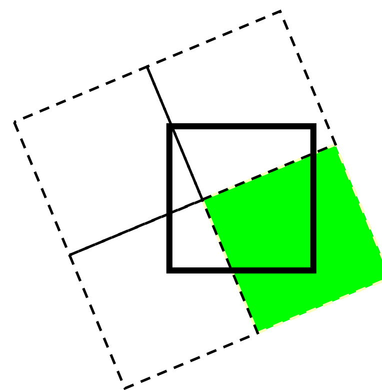
```
RescaleOp(1.5f, 0, null)
```

Rotations

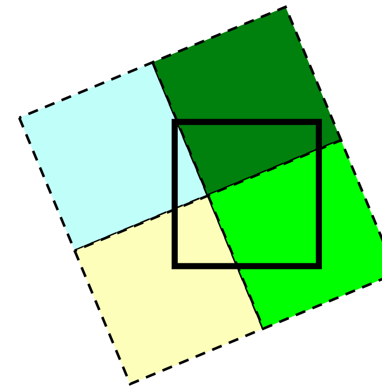




- Après rotation, une grille de points doit être affichée dans une autre grille. Quelle est la couleur d'un pixel ?
- Deux algorithmes proposés en Java:
  - ***nearest neighbor*** : la couleur est celle du pixel le plus proche
  - ***bilinear interpolation*** : la couleur est une combinaison des couleurs des quatre pixels source couvrant le pixel cible.



Nearest neighbor



Bilinear interpolation

Opérations

- Une rotation est une transformation affine
- par défaut, c'est le calcul au plus proche voisin
- le calcul bilinéaire est spécifié dans les **RenderingHints**

```
AffineTransform at = AffineTransform.getRotateInstance(Math.PI / 6);
RenderingHints rh = new RenderingHints(
        RenderingHints.KEY_INTERPOLATION,
        RenderingHints.VALUE_INTERPOLATION_BILINEAR);
BufferedImageOp op = new AffineTransformOp(at, rh));
BufferedImageOp op = new AffineTransformOp(at, null));
```

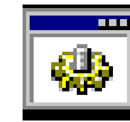
Manipulation directe par “Lookup”

- Une opération de “Lookup” remplace les intensités de couleurs par inspection (lookup) dans des tables.

```
LookupTable table = new ShortLookupTable(0, rouge);
BufferedImageOp op = new LookupOp(table, null);
```

- Les tables d’inspections peuvent être de type **`ByteLookupTable`** ou **`ShortLookupTable`**
- Dans les constructeurs,
  - le premier argument est un décalage dans la table,
  - le deuxième un tableau (d’entiers courts ou d’octets) à une ou deux dimensions.
    - une dimension : les mêmes changements pour les trois couleurs
    - deux dimensions : chaque composante RGB modifiée de façon indépendante

Exemples



- Inverser les intensités de toutes les couleurs

```
short[] inverser = new short[256];
for (int i = 0; i < 256; i++)
  inverser[i] = (short)(255 - i);
LookupTable table = new ShortLookupTable(0, inverser);
```

- Supprimer le rouge

```
short[] ident = new short[256];
short[] zero = new short[256];
for (int i = 0; i < 256; i++) {
  ident[i] = (short)i;
  zero[i] = (short)0;
}
short[][] sansRouge= { zero, ident, ident };
LookupTable table = new ShortLookupTable(0, sansRouge);
```

La fabrique des “lookup” (1)

```
class LookupFabrique {
  static short[] clair = new short[256];
  static short[] meilleurClair = new short[256];
  static short[] simplifie = new short[256];
  static short[] inverser = new short[256];
  static short[] ident = new short[256];
  static short[] zero = new short[256];
  static {
    for (int i = 0; i < 256; i++) {
      clair[i] = (short)(128 + i / 2);
      meilleurClair[i] = (short)(Math.sqrt((double)i / 255.0) * 255.0);
      simplifie[i] = (short)(i - (i % 32));
      inverser[i] = (short)(255 - i);
      ident[i] = (short)i;
      zero[i] = (short)0;
    }
  }
  static LookupOp createClair() {
      return new LookupOp(new ShortLookupTable(0, clair), null);}
  static LookupOp createMeilleurClair() {
      return new LookupOp( new ShortLookupTable(0, meilleurClair), null);}
  static LookupOp createSimplifie() {
      return new LookupOp( new ShortLookupTable(0, simplifie), null); }
  static LookupOp createInverser() {
      return new LookupOp(new ShortLookupTable(0, inverser), null);}
... }
```

La fabrique des “lookup” (2)

```
class LookupFabrique {//suite
...
  static short[][] rougeSeul = { inverser, ident, ident };
  static short[][] vertSeul = { ident, inverser, ident };
  static short[][] bleuSeul = { ident, ident, inverser };

  static LookupOp createInverserRouge() {
    return new LookupOp( new ShortLookupTable(0, rougeSeul), null);}

  static short[][] sansRouge= { zero, ident, ident };
  static short[][] sansVert = { ident, zero, ident };
  static short[][] sansBleu = { ident, ident, zero };

  static LookupOp createSansRouge() {
    return new LookupOp( new ShortLookupTable(0, sansRouge), null);}
}
```

Niveau de gris

- Pour passer en niveau de gris, on utilise une opération de conversion d'espace de couleurs `ColorConvertOp`
- Un exemple de construction est

```
new ColorConvertOp(ColorSpace.getInstance(ColorSpace.CS_GRAY), null));
```

- Les constructeurs sont:

```
ColorConvertOp(ColorSpace srcCspace, ColorSpace dstCspace, RenderingHints hints)
ColorConvertOp(ColorSpace cspace, RenderingHints hints)
ColorConvertOp(ICC_Profile[] profiles, RenderingHints hints)
ColorConvertOp(RenderingHints hints)
```

- Les deuxièmes et quatrièmes sont utiles pour les `BufferedImage,` les deux autres opèrent directement sur les raster.
- Un espace de couleur (comme RGB ou HSV) indique comment sont codées les couleurs.

Le panneau glissant

- En cliquant ou en glissant avec la souris, on “déplace” la ligne verticale et on découvre l’image transformée
- Ce panneau contient les deux images.
- A chaque événement, des rectangle de détourage (clipping) sont calculés pour n’afficher que la partie concernée.

```
public class SplitImagePanel extends JPanel {
  private BufferedImage image;   // source
  private BufferedImage trImage; // transformee
  private int splitX;
..}

..public void mousePressed(MouseEvent me) {
      splitX = me.getX(); repaint();
  }
..public void mouseDragged(MouseEvent me) {
      splitX = me.getX(); repaint();
  }
```

le dessin

```
public void paintComponent(Graphics g) {
   super.paintComponent(g);
   Graphics2D g2 = (Graphics2D)g;
   int width = getPreferredSize().width;
   int height = getPreferredSize().height;

   g2.clearRect(0, 0, width, height);
   g2.setClip(splitX, 0, width - splitX, height);
   g2.drawImage(getImage(), 0, 0, null);

   if (splitX == 0 || trImage == null) return;

   g2.setClip(0, 0, splitX, height);
   g2.drawImage(trImage, 0, 0, null);

   Line2D splitLine = new Line2D.Float(splitX, 0, splitX, height);
   g2.setClip(null);
   g2.setColor(Color.white);
   g2.draw(splitLine);
}
```