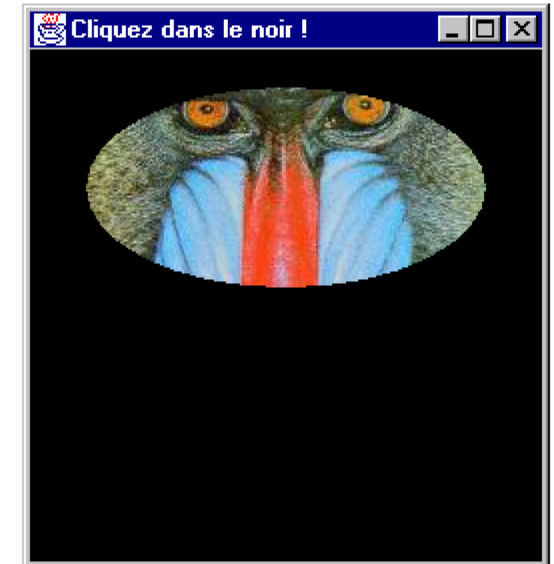
www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Dessin Java et Java 2D

- Java 2D et Swing
- Graphics
- repaint(), et en Swing
- Composants de base
- Chaîne de traitement
- Les formes (Shape)
- Les courbes, les aires
- Transformations affines

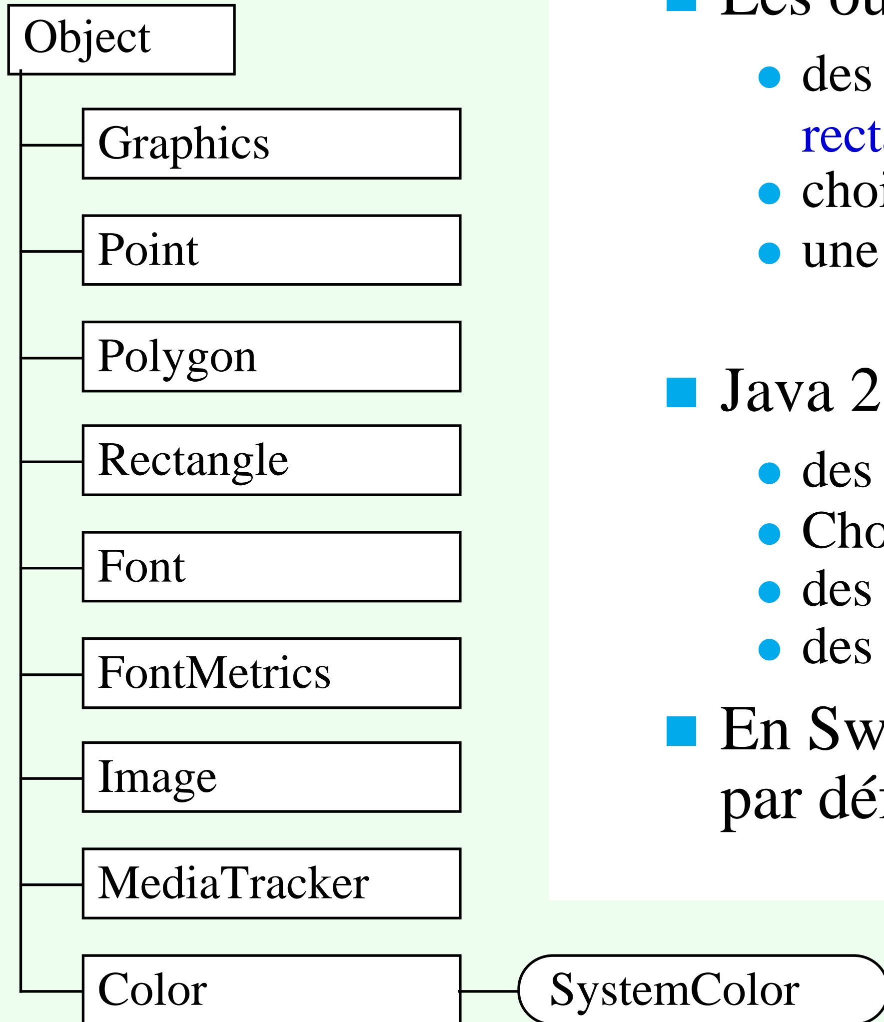


2D et Swing : deux avantages

- Pour les composants : AWT et Swing
- Pour l'affichage : Graphics et Graphics2D

	Graphics	Graphics2D
AWT	Affichage de base: Texte, formes géométriques simples	Concepts nouveaux: Shape, Transformations, Path
Swing	Affichage sophistiqué: double buffering par défaut, calcul de la zone de rafraîchissement	Concepts nouveaux + affichage sophistiqué

Le dessin de base



- Les outils de dessin sont assez rudimentaires :
  - des méthodes **draw**`*()` et **fill**`*()` pour lignes, rectangles, ovales, polygone;
  - choix de deux modes de dessin : **direct** ou **xor**;
  - une zone de découpe (clipping) rectangulaire.
- Java 2 propose des possibilités très sophistiquées:
  - des méthods **draw(Shape)** et **fill(Shape)**
  - Choix des 8 modes de dessin
  - des zones de découpe arbitraires (en principe)
  - des transformations géométriques complexes
- En Swing, le “double buffering” est automatique par défaut.

Contexte graphique

- L'outil de dessin est le ***contexte graphique***, objet de la classe **`Graphics`**. Il encapsule l'information nécessaire, sous forme d'***état graphique***.

  Celui-ci comporte
  - la zone de dessin (le ***composant***), pour les méthodes **`draw*()`** et **`fill*()`**
  - une éventuelle translation d'origine
  - le rectangle de découpe (***clipping***)
  - la couleur courante
  - la fonte courante
  - l'opération de dessin (simple ou xor)
  - la couleur du xor, s'il y a lieu.
- Chaque composant peut accéder implicitement et explicitement à un contexte graphique.

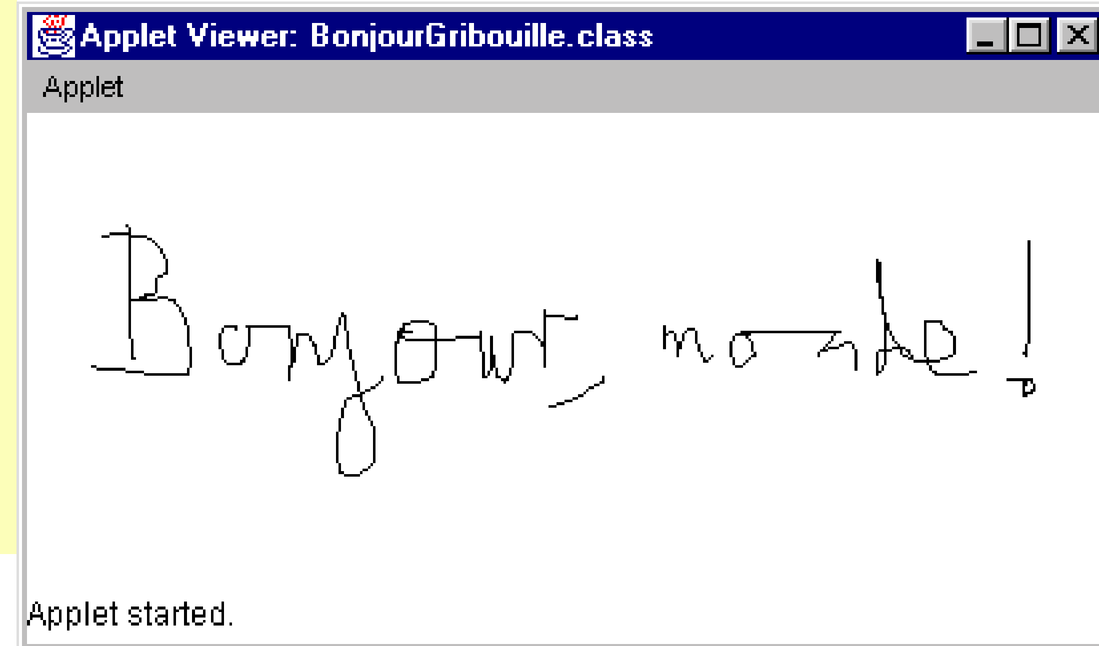
Obtenir un contexte graphique

- On obtient un *contexte graphique*
  - ***implicitement***, dans une méthode `paint()` ou `update()`: AWT construit un contexte graphique passé en paramètre,
  - ***explicitement***, dans un composant ou dans une image, par `getGraphics()`,
  - ***explicitement*** encore, en copiant un objet `Graphics` existant.
- Un contexte graphique utilise des ressources systèmes. L'acquisition explicite doit être accompagnée, in fine, par une *libération explicite* au moyen de `dispose()`.
- L'acquisition explicite d'un contexte graphique est - dit-on - signe d'une programmation maladroite.

Un exemple

```
public class BonjourGribouille
  extends Applet {
  int xd, yd;

  public void init() {
    addMouseListener(new Appuyeur());
    addMouseMotionListener(new Dragueur());
  }
  class Appuyeur extends MouseAdapter {
    public void mousePressed(MouseEvent e) {
      xd = e.getX(); yd = e.getY();
    }
  }
  class Dragueur extends MouseMotionAdapter {
    public void mouseDragged(MouseEvent e) {
      int x = e.getX(), y = e.getY();
      Graphics g = getGraphics();
      g.drawLine(xd, yd, x, y);
      xd = x; yd = y;
      g.dispose();
    }
  }
}
```

- Accès *explicite* à un contexte graphique.
- A chaque **getGraphics()**, un *nouveau* contexte est fourni, ne connaissant rien du précédent.



Sans appel explicite : AWT seulement

- Fait appel à la ***triplette magique***

  repaint
  update
  paint

- **repaint()** demande un rafraîchissement. Appelle **update()**, en lui fournissant un contexte graphique.
- **update()** par défaut efface le dessin et appelle **paint()**.
- **paint()** par défaut ne fait rien.
- ***Ici***
  - **repaint()** appelle **update();**
  - **update()** n'efface pas;
  - **paint()** trace la ligne.

```
public class BonjourGribouille2
  extends Applet {
  int xd, yd, x, y;

  public void init() { idem }

  public void update(Graphics g) {
    paint(g);
  }
  public void paint(Graphics g) {
    g.drawLine(xd, yd, x, y);
    xd = x; yd = y;
  }
  class Appuyeur extends MouseAdapter {
    public void mousePressed(MouseEvent e) {
      xd = e.getX(); yd = e.getY();
    }
  }
  class Dragueur extends MouseMotionAdapter {
    public void mouseDragged(MouseEvent e) {
      x = e.getX(); y = e.getY();
      repaint();
    }
  }
}
```

repaint()

- C'est la ***méthode par excellence*** pour rafraîchir un affichage !

```
Code simplifié de Component.java
public void Component.update(Graphics g) {
   g.setColor(getBackground());
   g.fillRect(0,0, width, height);
   g.setColor(getForeGround());
   paint(g);
}
public void Component.paint(Graphics g){}
```

- **`repaint()`** poste un appel à **`update().`** Plusieurs appels peuvent être groupés.
- **`update()`** effectue les opérations suivantes:
  - efface le composant en le remplissant avec la couleur de fond
  - définit la couleur du contexte à la couleur de dessin
  - appelle **`paint().`**
- **`paint()`** ne fait rien par défaut.
- **`repaint()`** est appelé automatiquement à la retaille d'une fenêtre.
- Pour dessiner, on redéfinit **`paint()`** ou **`update()`** (ou les deux).

repaint() en Swing

- Reste la ***méthode par excellence*** pour rafraîchir un affichage !

```
Code simplifié de ComponentUI.java
public void update(Graphics g, JComponent c) {
  if (c.isOpaque()) {
    g.setColor(c.getBackground());
    g.fillRect(0,0, c.getWidth(), c.getHeight());
  }
  paint(g, c);
}
```

- **`repaint()`** poste un appel à **`update()`.** Plusieurs appels peuvent être groupés.
- **`update()`** appelle **`paint()`.**
- **`paint()`** appelle successivement
  - **`paintComponent()`** pour le dessin (le **`paint()`** en AWT)
  - **`paintBorder()`**
  - **`paintChildren()`.**
- **`paintComponent()`** par défaut appelle **`ComponentUI.update()`** qui efface et redessine le fond ***si le composant est opaque*** (**`JPanel`** l'est par défaut).
- Pour dessiner, on redéfinit **`paintComponent()`** et il est utile d'appeler **`super.paintComponent()`.**

Lignes, rectangles, ovales

méthode	description
`drawLine()`	trace une ligne
`drawRect()`	trace un rectangle
`drawOvale()`	trace une ovale
`drawArc()`	trace un arc
`drawRoundRect()`	rectangle à bords arrondis
`draw3DRect()`	rectangle ombré
`drawPolygone()`	un polygone fermé
`drawPolyline()`	une ligne polygonale

méthode	description
`fillArc()`	remplit un arc
`fillRect()`	remplit un rectangle
`fillOvale()`	remplit une ovale
`fillRoundRect()`	rectangle à bords arrondis
`fill3DRect()`	rectangle ombré
`fillPolygone()`	remplit un polygone fermé

- Le contour est dessiné par **`draw`**, l'intérieur est rempli par **`fill`**.
- Tracer une ligne, une ovale, un rectangle est sans surprise.
- L'interaction **`draw`** et **`fill`** est usuelle : si l'on veut voir le contour, il faut le tracer ***après***.

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Polygônes

- Un **polygône** est une ligne polygonale fermée, donnée par la suite de ses points. Les premier et dernier points sont joints.
- Deux constructeurs, et possibilité d'ajouter un point.
- Remplissage selon la "even-odd rule".

Constructeurs

```
Polygon()
Polygon(int[] xp, int[] yp, int np)
```

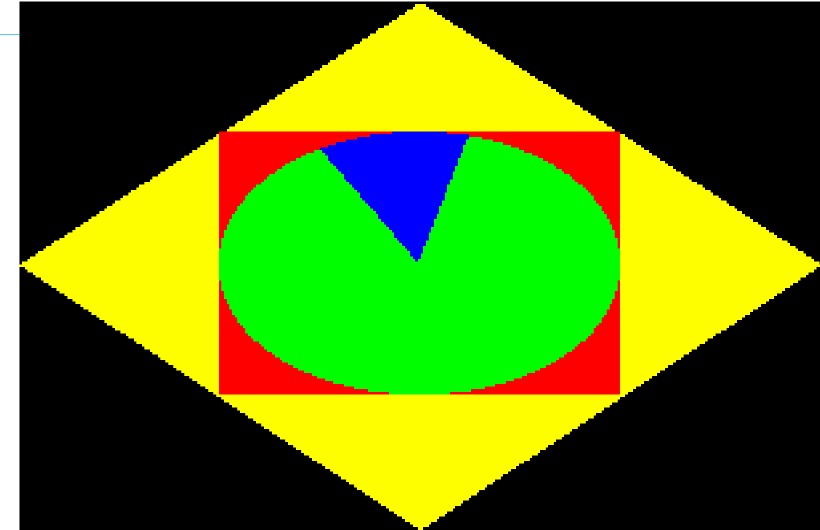
Données

```
npoints
xpoints
ypoints
```

Méthodes

```
addPoint(int x, int y)
contains(Point p)
contains(int x, int y)
getBounds()
translate(int dx, int dy)
```

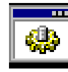
Exemple

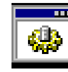


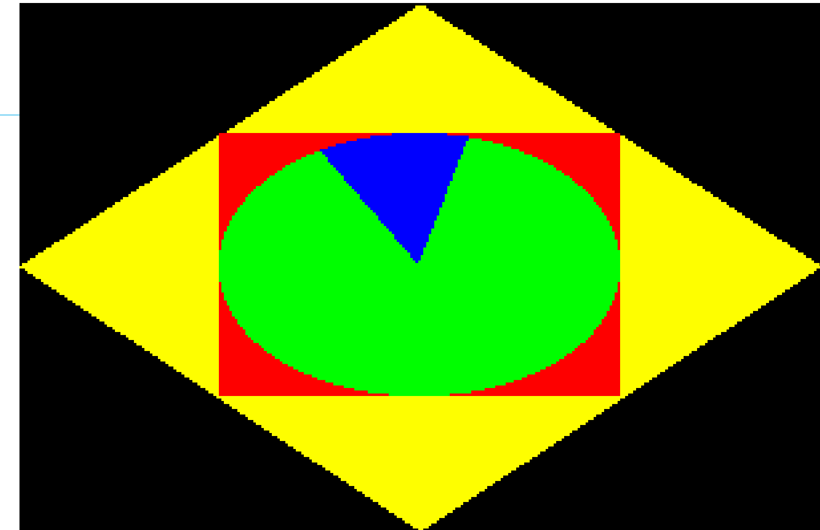
```java
public void paint(Graphics g) {
  int largeur = getSize().width;
  int hauteur = getSize().height;
  int dl = largeur/2, dh = hauteur/2;
  int [] polx = { 0, dl, largeur, dl};
  int [] poly = {dh, 0, dh, hauteur};
  Polygon pol = new Polygon(polx,poly,4);

  g.setColor(Color.black);
  g.fillRect(0,0,largeur,hauteur);
  g.setColor( Color.yellow);
  g.fillPolygon(pol);
  g.setColor( Color.red);
  g.fillRect(dl/2, dh/2, dl,dh);
  g.setColor( Color.green);
  g.fillOval(dl/2, dh/2, dl,dh);
  g.setColor( Color.blue);
  g.fillArc(dl/2, dh/2, dl, dh, th, del);
}
```

```java
public class Losange extends Applet{
  int th = 45, del =45;
  public void init(){
    addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent e){
        th = (th +10)%360;
        repaint();
      }
    });
  }
  public void paint(Graphics g) {...}
}
```

Et en Swing







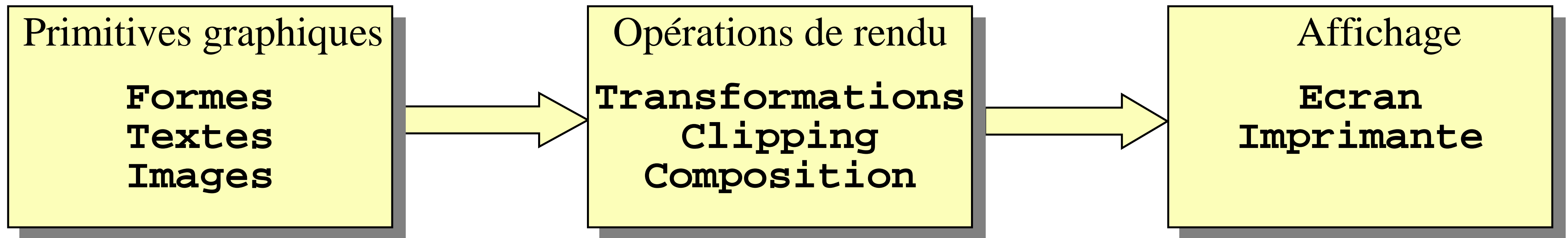
```
class Dessin extends JPanel {
  int theta = 45, del =45;
  public void paintComponent(Graphics g) {
    int largeur = getSize().width;
    int hauteur = getSize().height;
    int dl = largeur/2, dh = hauteur/2;
    int [] polx = { 0, dl, largeur, dl};
    int [] poly = {dh, 0, dh, hauteur};
    Polygon pol = new Polygon(polx,poly,4);
    g.setColor(Color.black);
    g.fillRect(0,0,largeur,hauteur);
    g.setColor( Color.yellow);
    g.fillPolygon(pol);
    g.setColor( Color.red);
    g.fillRect(dl/2, dh/2, dl,dh);
    g.setColor( Color.green);
    g.fillOval(dl/2, dh/2, dl,dh);
    g.setColor( Color.blue);
    g.fillArc(dl/2, dh/2, dl, dh,theta, del);
  }
...
```

```
public class Losange extends JApplet  {
  public void init(){
        setContentPane(new Dessin());
  }
}
```

```
...
  public Dessin() {
   addMouseListener( new MouseAdapter() {
     public void mousePressed(MouseEvent e){
       theta = (theta + 10)%360;
       repaint();
     }
   });
  }
}
```

Chaîne de traitement 2D

- Le processus de traitement est en plusieurs étapes
  - déterminer ce qui doit être affiché (formes, textes, images)
  - appliquer les transformations géométriques et le clipping
  - déterminer la couleur
  - combiner avec ce qui se trouve sur la surface
- Pour chacune de ces étapes, des multiples possibilités existent.



Détails

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

- On obtient un objet **Graphics2D** par conversion

```
public void paintComponent(Graphics g) {
  Graphics2D g2 = (Graphics2D) g;
  ...
```

- On améliore l'affichage par antialiasing etc

```
RenderingHints hints = ...;
g2.setRenderingHints(hints);
```

- On choisit l'outil de dessin au trait (contours)

```
Stroke stroke = ...;
g2.setStroke(stroke);
```

Détails (suite)

- On choisit l’outil de remplissage (couleur, dégradé, motif)

```
Paint paint = ...;
g2.setPaint(paint);
```

- On définit la forme de découpage

```
Shape clip = ...;
g2.setClip(clip);
```

- On définit une transformation géométrique entre l’espace utilisateur et espace d’écran

```
AffineTransfrom at = ...;
g2.transform(at);
```

Détails (fin)

- On ompose le dessin résultant avec le dessin existant

```
Composite composite = ...;
g2.setComposite(composite);
```

- On définit la forme à dessiner

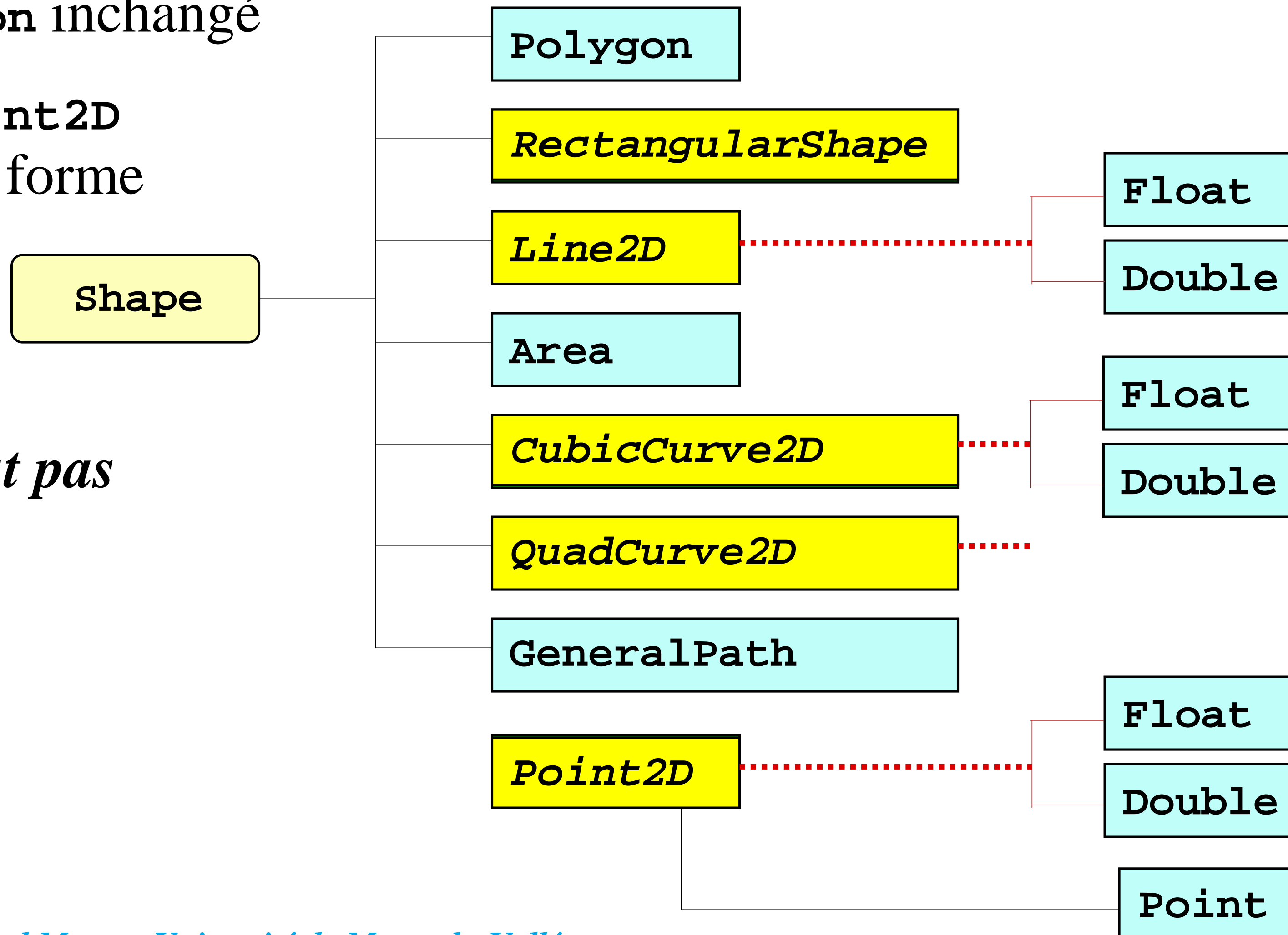
```
Shape dessin = ...;
```

- On l'affiche

```
g2.fill(dessin);
g2.draw(dessin);
```

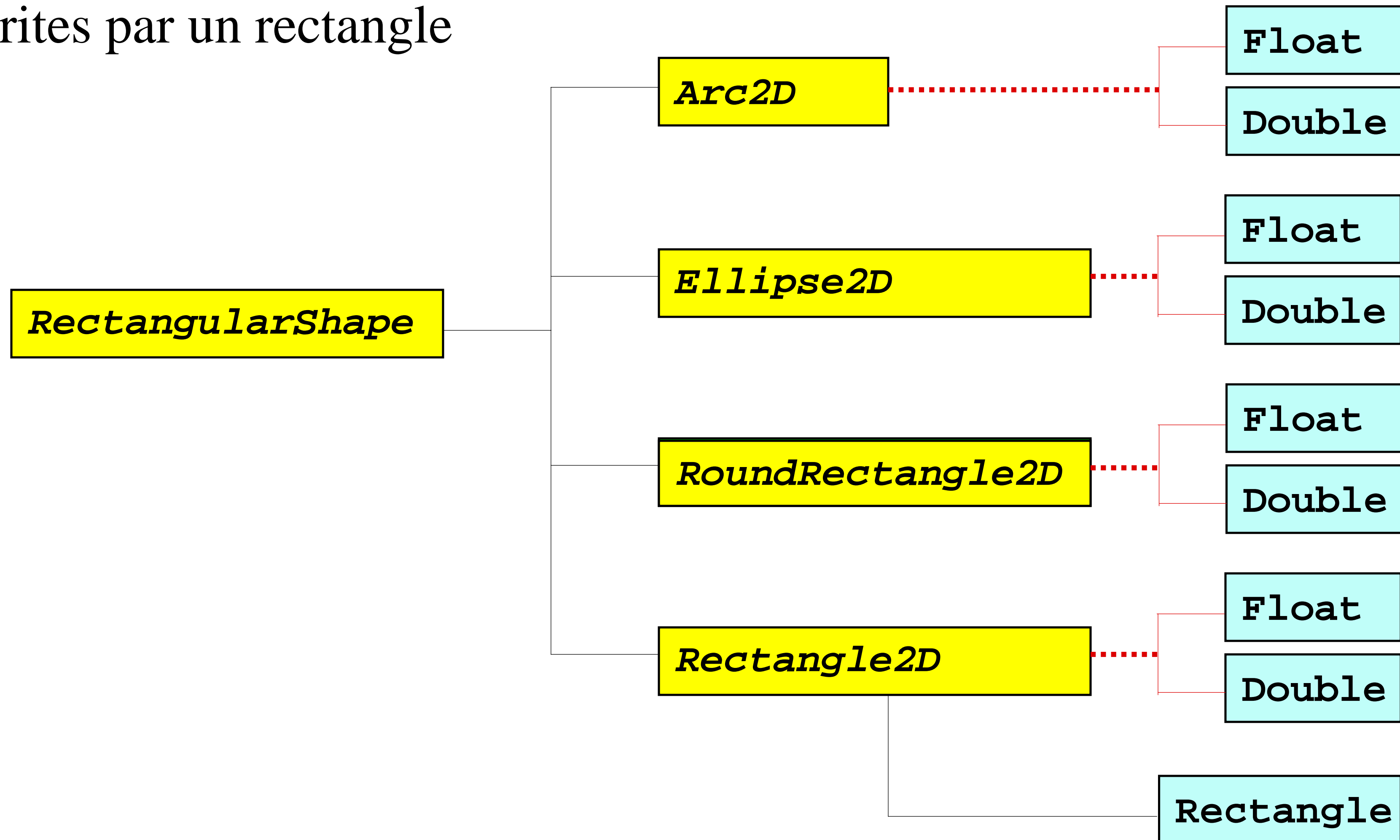
Les formes (1)

- Dans le paquetage **java.awt.geom**
- Mais **Polygon** inchangé
- La classe **Point2D** n'est pas une forme
- ***un point n'est pas tracé.***



Les formes (2)

- **`RectangularShape`** est la classe abstraite de base des formes qui sont décrites par un rectangle



Usage avec Graphics2D

- La classe `Graphics2D` est une classe dérivée de `Graphics`
- La méthode `JComponent.paintComponent(Graphics g)` reçoit en fait un `Graphics2D`.De même pour `paint().`
- On convertit `g` par

```
Graphics2D g2  = (Graphics2D) g;
```

- On utilise des méthodes

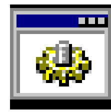
```
fill(Shape s)
draw(Shape s)
```

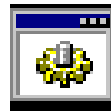
- Exemple

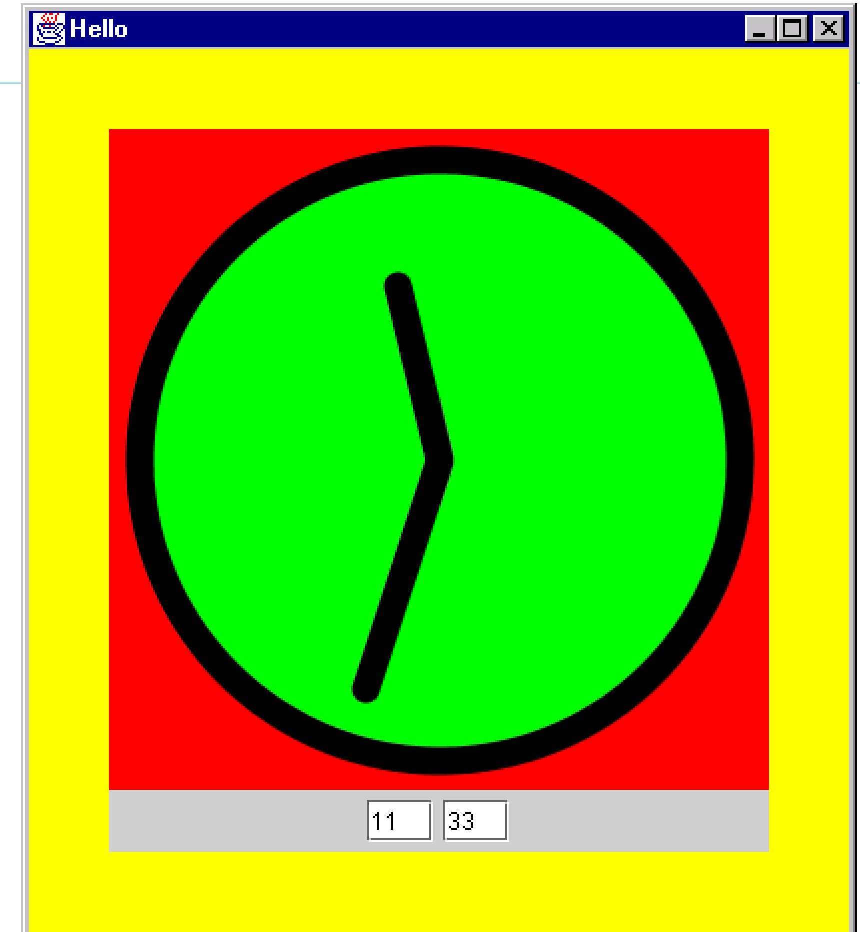
```
g2.fill(new Rectangle2D.Double(x, y, 100, 100));
```

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Traits


Horloge.bat


StrokeTest.bat



- Les traits se dessinent avec une plume de la l’interface Stroke, implémentée par BasicStroke.
- Les attributs sont
  - l’épaisseur (width)
  - fins de traits (end caps)
    CAP_BUTT, CAP_ROUND, CAP_SQUARE
  - lien entre traits (join caps)
    JOIN_BEVEL, JOIN_MITER, JOIN_ROUND
  - pointillé (dash)
- Par défaut
  - trait continue d’épaisseur **1, CAP_SQUARE, JOIN_MITER**, miter limit 10

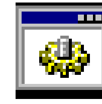
```
g2.setStroke(new BasicStroke(14,
    BasicStroke.CAP_ROUND, BasicStroke.JOIN_BEVEL));
```

Détails

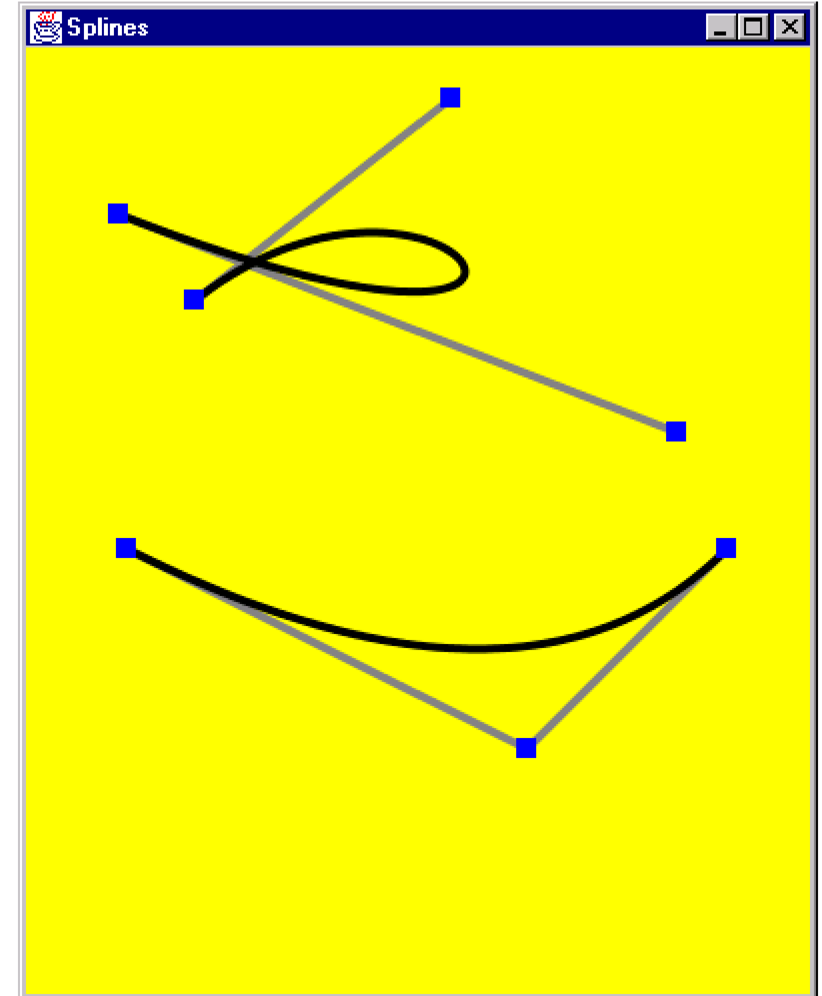
- Cet exemple contient l'illustration de plusieurs aspects

```
public void paintComponent(Graphics g) {
  Graphics2D g2 = (Graphics2D) g;
  g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
    RenderingHints.VALUE_ANTIALIAS_ON);
  g2.setRenderingHint(RenderingHints.KEY_RENDERING,
    RenderingHints.VALUE_RENDER_QUALITY);
  super.paintComponent(g2);
  g2.translate(getWidth()/2,getHeight()/2);
  g2.setColor(Color.green);
  g2.fillOval(-taille/2,-taille/2, taille, taille);
  g2.setColor(Color.black);
  g2.setStroke(new BasicStroke(14,
    BasicStroke.CAP_ROUND, BasicStroke.JOIN_BEVEL));
  g2.drawOval(-taille/2,-taille/2, taille, taille);
  double angleH = 2*Math.PI* (minutes - 3*60) / (12*60);
  double angleM = 2*Math.PI* (minutes - 15) / 60;
  GeneralPath gp = new GeneralPath();
  gp.moveTo((int)(0.3*taille*Math.cos(angleH)),
            (int)(0.3*taille*Math.sin(angleH)));
  gp.lineTo(0,0);
  gp.lineTo((int)(0.4*taille*Math.cos(angleM)),
            (int)(0.4*taille*Math.sin(angleM)));
  g2.draw(gp);
}
```

Courbes de Bézier



- Elles sont quadratiques ou cubique.
- Elles ont trois ou quatre points de contrôle, dont deux sont des extrémités.
- La courbe est contenue dans le polygône de contrôle formé des trois ou quatre points.



Construction

- Un constructeur de courbe cubique, en double:

```
CubicCurve2D.Double(double x1, double y1, double ctrlx1, double ctrly1,
   double ctrlx2, double ctrly2, double x2, double y2)
```

- Variantes utiles

```
CubicCurve2D.Double()  // initialisée à zéro
CubicCurve2D.setCurve(double[] coords, int offset) // affecte les valeurs
CubicCurve2D.setCurve(Point2D[] pts, int offset)
CubicCurve2D.setCurve(Point2D p1, Point2D cp1, Point2D cp2, Point2D p2)
```

- les mêmes valent pour les courbes quadratiques.

Exemple

- Données

```
protected Point2D[] points = new Point2D[7];
points[0] = new Point2D.Double(50, 100);
...
points[6] = new Point2D.Double(350, 250);
Line2D tangent1 = new Line2D.Double();
Line2D tangent2 = new Line2D.Double();

CubicCurve2D c = new CubicCurve2D.Double();
QuadCurve2D q = new QuadCurve2D.Double();
```

- Dessin

```
public void paintComponent(Graphics g) {
  Graphics2D g2 = (Graphics2D)g;
  super.paintComponent(g);
  g2.setStroke(new BasicStroke(4));

  tangent1.setLine(points[0], points[1]);
  tangent2.setLine(points[2], points[3]);
  g2.setPaint(Color.gray);
  g2.draw(tangent1);
  g2.draw(tangent2);

  c.setCurve(points, 0);
  g2.setPaint(Color.black);
  g2.draw(c);
  ...
}
```

Exemple : fin

- Et les petits carrés. Donnée

```
Rectangle2D squareRect = new Rectangle2D.Double();
```

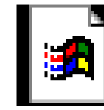
- Dessin: on ne dessine pas un point ...

```
public void paintComponent(Graphics g) {
  ...
  for (int i = 0; i < points.length; i++) {
    if (points[i] == selectedPoint)
      g2.setPaint(Color.red);
    else
     g2.setPaint(Color.blue);
  g2.fill(getSquare(points[i]));
  }
}
```

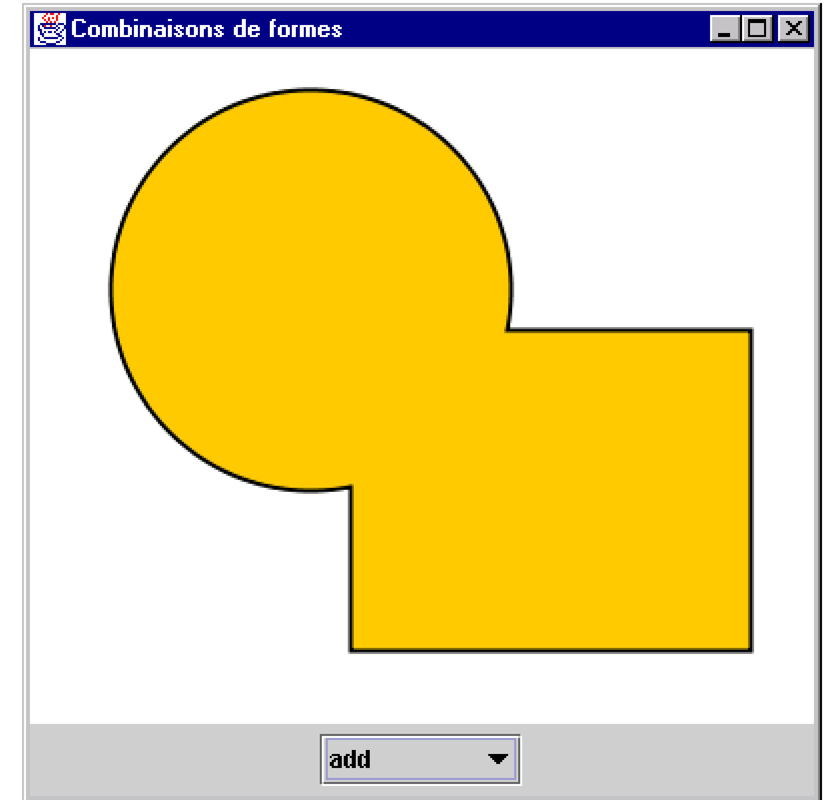
- Méthode auxiliaire

```
Shape getSquare(Point2D p) {
  int side = 10;
  squareRect.setRect(p.getX()-side/2,
    p.getY()-side/2, side, side);
  return squareRect;
}
```

Aires


CombiningShapes.bat

- Formes que l'on peut composer par des opérations booléennes (***constructive solid geometry*** en 2D)
- Toute forme est un composant
- Les opérations sont
  - add
  - subtract
  - intersect
  - exclusiveor



Aires : opérateurs

```
public void paintComponent(Graphics g) {
  Graphics2D g2 = (Graphics2D)g;
  ...
  Area areaOne = new Area(ellipse);
  Area areaTwo = new Area(rectangle);
  if (option.equals("add")) areaOne.add(areaTwo);
  else if (option.equals("intersection")) areaOne.intersect(areaTwo);
  else if (option.equals("subtract")) areaOne.subtract(areaTwo);
  else if (option.equals("exclusive or")) areaOne.exclusiveOr(areaTwo);
  g2.setPaint(Color.orange);
  g2.fill(areaOne);
  g2.setPaint(Color.black);
  g2.draw(areaOne);
}
```

Découpage

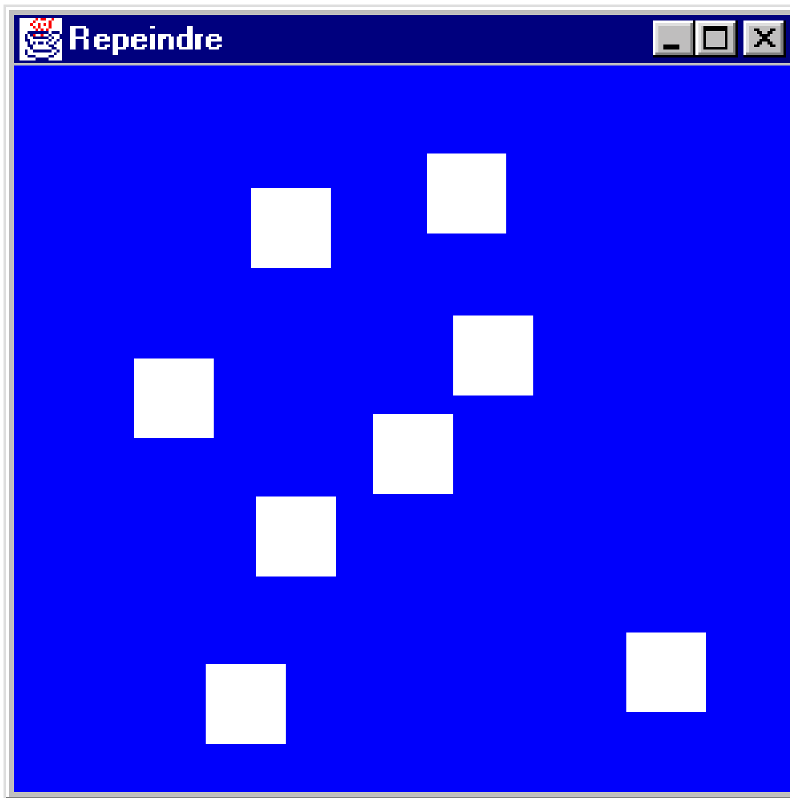
- On peut restreindre la zone à (re)dessiner de deux manières
  - par la définition d'une ***forme de découpe*** (clipping)
  - par la spécification d'un rectangle de rafraîchissement dans la méthode **`repaint()`.**
- Un contexte graphique contient un rectangle de découpe
  - initialement toute la zone de dessin
  - modifiable par **`setClip()`** le rectangle ne peut que diminuer
- La méthode **`repaint()`** de **`Component`** peut prendre en argument un rectangle, et limiter ainsi l'action à ce rectangle.

```
repaint()
repaint(long tm)
repaint(int x, int y, int width, int height)
repaint(long tm, int x, int y, int width, int height)
```

Un exemple

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

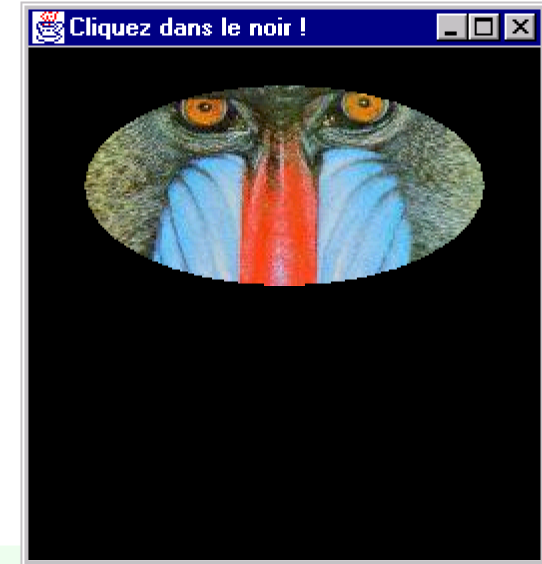
- Dans le rectangle bleu, on "repeint" des petits carrés, faisant ainsi apparaître le fond blanc.




```
class Repeindre extends Frame  {
  Repeindre() {
    setTitle("Repeindre");
    addMouseListener(new Reveleur());
    setSize(300,300);
    setVisible(true);
    Graphics g = getGraphics();
    g.setColor(Color.blue);
    g.fillRect(0, 0,
      getSize().width, getSize().height);
    g.dispose();
  }

  class Reveleur extends MouseAdapter {
    public void mousePressed(MouseEvent e) {
      repaint(e.getX(), e.getY(), 30, 30);
    }
  }

  public static void main(String[] args) {
    new Repeindre();
  }
}
```

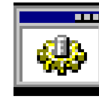
Un deuxième exemple

- On choisit une ellipse comme région de découpage. On dessine une image qui paraît être partiellement révélée.
- L'usage de la propriété `opaque` donne d'autres effets.




```
class ImagePanel extends JPanel  {
  private Image image;
  private Ellipse2D oeil = new Ellipse2D.Double();
  private boolean pressed = false;

  public ImagePanel() {
    image = new ImageIcon("mandrill.jpg").getImage();
    setPreferredSize(
      new Dimension(image.getWidth(this),image.getHeight(this)));
      ...
    setBackground(Color.black);
  }
  ...
```

L'affichage



- On ne peint que la partie qui est visible à travers l'oeil

```
public void paintComponent(Graphics g) {
  super.paintComponent(g);
  int width = getSize().width;
  int height = getSize().height;
  if (pressed) {
    g.setClip(oeil);
    g.drawImage(image, 0, 0, this);
  }
}
```

- et on modifie l'ellipse en fonction de position de la souris

```
public void mouseDragged(MouseEvent e) {
  oeil.setFrameFromCenter(e.getX(), e.getY(), e.getX() + 100,  e.getY() + 50);
  repaint();
}
```

Découverte

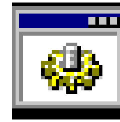
- La modification de l'opacité empêche le rafraîchissement de la fenêtre : c'est comme dans un jeu où on gratte.

```
public void mousePressed(MouseEvent e) {
  pressed = true;
  oeil.setFrameFromCenter(e.getX(), e.getY(), e.getX() + 100,  e.getY() + 50);
  if ((e.getModifiers() & MouseEvent.BUTTON3_MASK)!=0)
    setOpaque(!isOpaque());
  repaint();
}
```

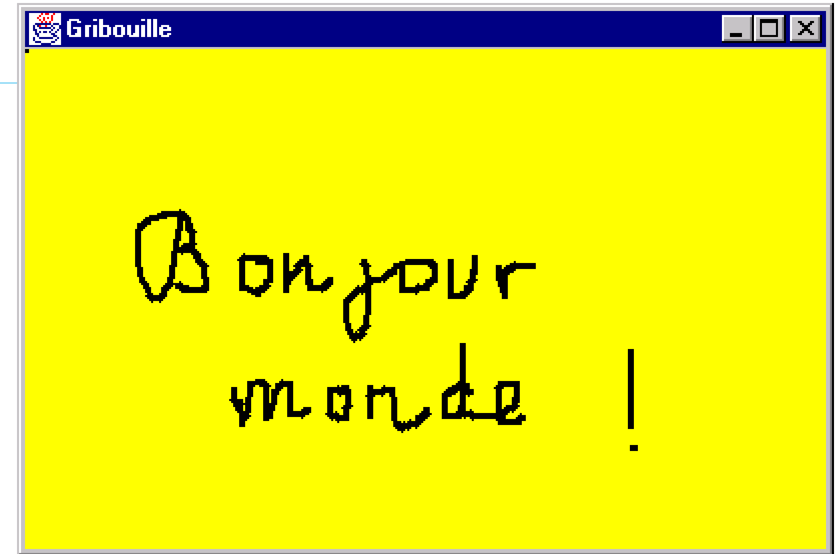
Couleurs

- La classe **Color** permet de gérer les couleurs. Constantes
  black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow.
- Constructeurs rgb, p.ex. **Color(int r, int g, int b);**
- Conversion **RGBtoHSB** (hue, saturation, brightness) et vice-versa.
- La classe dérivée **SystemColor** contient des noms symboliques pour les couleurs du système: contrôle, fenêtre active, menu, ombre (inspiré de Windows).
- Le coefficient alpha indique la transparence (0 = opaque, 1 = transparent)

Gribouilleur



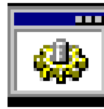
JGribouille.bat

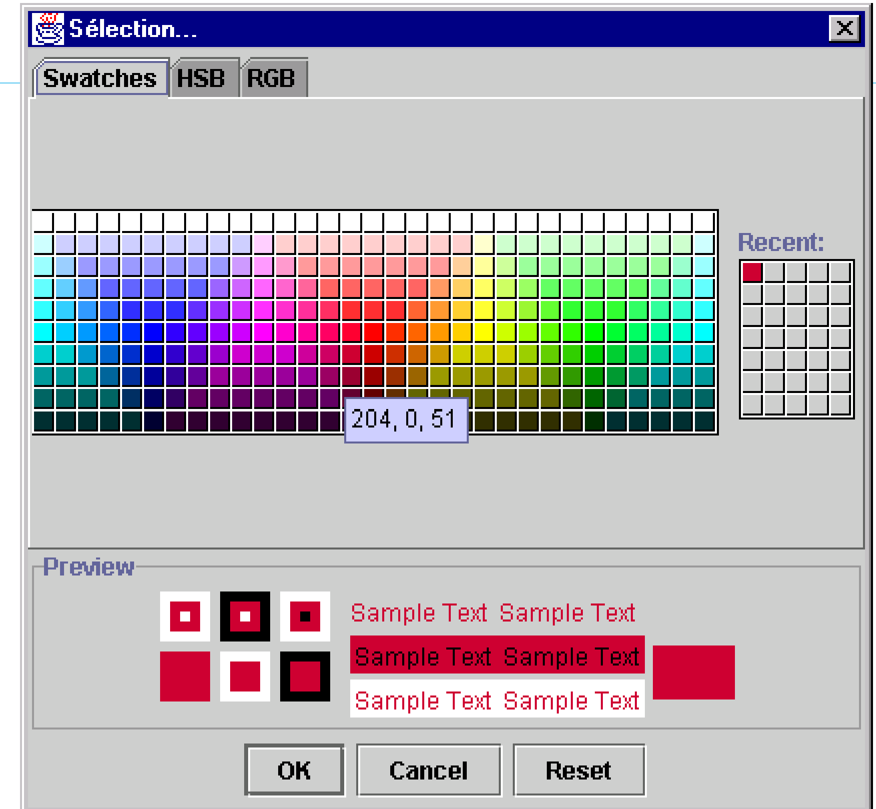



```
public class JGribouille {
  public static void main(String[] args) {
    JFrame f = new JFrame("Gribouille");
    f.setContentPane(new Gribouilleur());
    f.pack();
    f.setVisible(true);
    f.setBackground(Color.yellow);
    f.addWindowListener(new Fermeur());
  }
}

class Gribouilleur extends JPanel {
  int xd, yd, x, y;
  Gribouilleur() {
    setPreferredSize(new Dimension(400,250));
    addMouseListener(new Appuyeur());
    addMouseMotionListener(new Dragueur());
    setOpaque(false);
  }
  public void paintComponent(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    g2.setStroke(new BasicStroke(3));
    g2.drawLine(xd, yd, x, y);
    xd = x; yd = y;
  }...
```

```
...
class Appuyeur extends MouseAdapter {
  public void mousePressed(MouseEvent e) {
    xd = e.getX(); yd = e.getY();
  }
}
class Dragueur extends MouseMotionAdapter {
  public void mouseDragged(MouseEvent e) {
    x = e.getX(); y = e.getY();
    repaint();
  }
}
}
```

ColorChooser


ChoixCouleur.bat

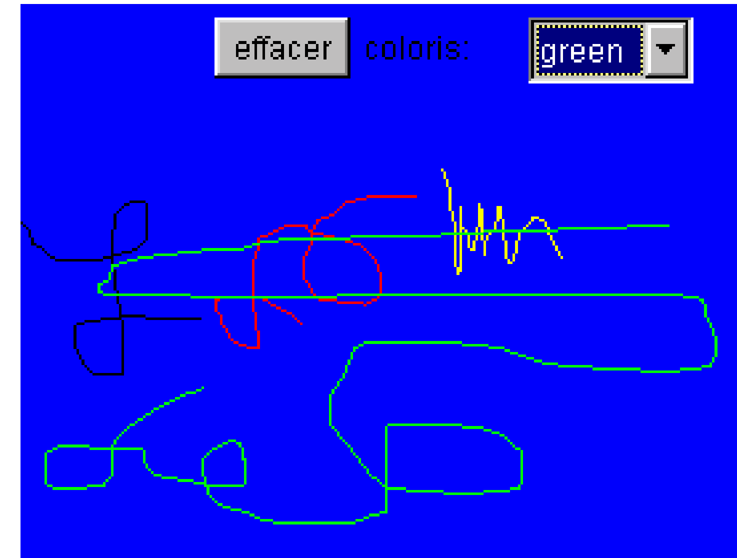


- Un composant de sélection de couleur est fourni. Il opère en plusieurs modèles par défaut, et peut être configuré.
- Ici, initialisé avec couleur de fond.
- Le bouton qui appelle le sélectionneur est dans un panneau.

```
JButton colorButton = new JButton("Couleurs...");
colorButton.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
    Color c = JColorChooser.showDialog(Panneau.this, "Sélection...", getBackground());
    if (c != null) setBackground(c);
  }
});
```

Exemple

```
public class Gribouille3 extends Applet {
  int xd, yd, x, y;
  Color c = Color.black;
  Button nettoyer;
  Choice couleurs;
public Gribouille3() {
  setBackground(Color.blue);
  nettoyer = new Button("effacer");
  nettoyer.setForeground(Color.black);
  nettoyer.setBackground(Color.lightGray);
  couleurs = new Choice();
  couleurs.addItem("black");
  couleurs.addItem("red");
  couleurs.addItem("yellow");
  couleurs.addItem("green");
  couleurs.setForeground(Color.black);
  couleurs.setBackground(Color.lightGray);
}
public void init() {
  add(nettoyer);
  add(new Label("coloris: "));
  add(couleurs);
  addMouseListener(new Appuyeur());
  addMouseMotionListener(new Dragueur());
  nettoyer.addActionListener(new Nettoyeur());
  couleurs.addItemListener(new Coloreur());
}
```




```
public void update(Graphics g) {
  g.setColor(c);
  paint(g);
}
public void paint(Graphics g) {
  g.drawLine(xd, yd, x, y);
  xd = x; yd = y;
}
```

Exemple (suite)

- Le *Nettoyeur* efface tout

```
class Nettoyeur implements  ActionListener {
  public void actionPerformed(ActionEvent e) {
    Graphics g = getGraphics();
   g.clearRect(0,0,getSize().width,
     getSize().height);
   g.dispose();
  }
}
```

- L'*Appuyeur* relève la position

```
class Appuyeur extends MouseAdapter {
  public void mousePressed(MouseEvent e) {
  xd = e.getX(); yd = e.getY();
  }
}
```

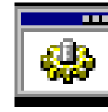
- Le *Dragueur* relève la nouvelle position et demande le dessin

```
class Dragueur extends MouseMotionAdapter {
  public void mouseDragged(MouseEvent e) {
    x = e.getX(); y = e.getY();
    repaint();
  }
}
```

- Le *Coloreur* relève la nouvelle couleur

```
class Coloreur implements ItemListener {
  public void itemStateChanged(ItemEvent e) {
    String a = (String) e.getItem();
      if (a.equals("black")) c = Color.black;
      else if (a.equals("red")) c = Color.red;
      else if (a.equals("yellow"))
        c = Color.yellow;
      else if (a.equals("green"))
        c = Color.green;
      else c = Color.pink;
  }
}
```

Dégradés et textures



- **GradientPaint** et **TexturePaint** implémentent **Paint**
- **GradientPaint** crée un dégradé entre deux couleurs données en deux points

```
Paint paint = new GradientPaint(0, 0, Color.red,
  (float)getWidth()/2, (float)getHeight()/2, Color.blue);
g2.setPaint(paint);
g2.fill(ellipse);
```

- **TexturePaint** répète une image plaquée dans un rectangle jusqu'à remplir la forme.

```
Rectangle2D anchor = new Rectangle2D.Double(0, 0,
  4 * bufferedImage.getWidth(), 4 * bufferedImage.getHeight());
Paint paint = new TexturePaint(bufferedImage, anchor);
```

La composition

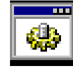
www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

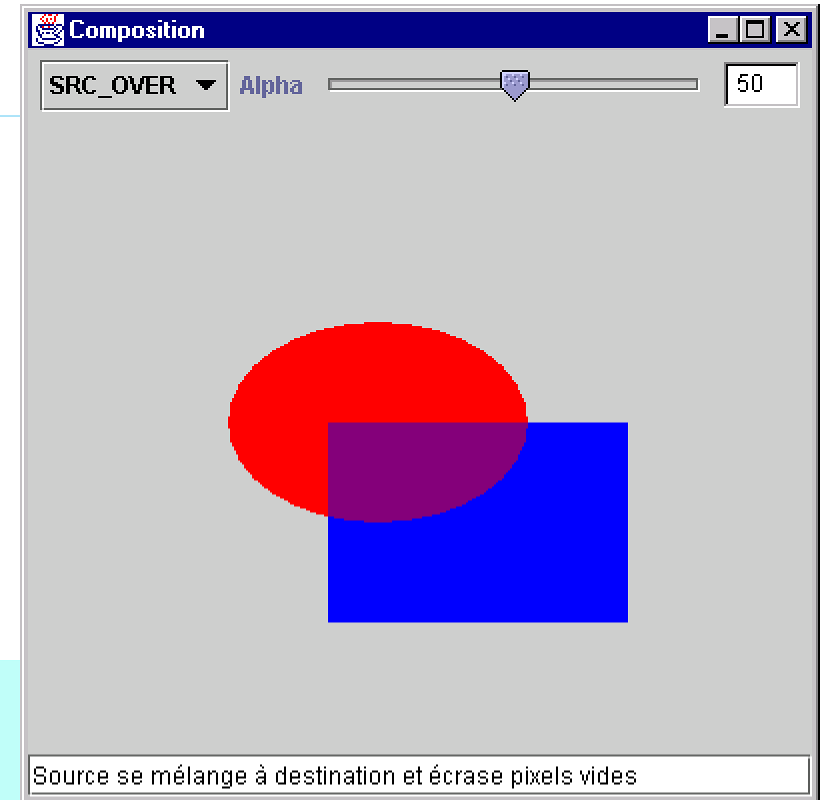
- Il y a 8 modes de composition de l'image construite avec l'image existante, numérotées par des constantes de la classe `AlphaComposite`.
- Le coefficient alpha ne change pas ces modes, mais atténue seulement l'impact de l'image construite.
- **s** = alpha de source, **d** = alpha de destination, le coefficient final du mélange est ***s(1-d)*** ou ***d(1-s).***
- On choisit le style de composition par

```
Composite composite = AlphaComposite.getInstance(rule, alpha);
g2.setComposite(composite);
```

- Encore faut-il que l'écran accepte une "couche alpha". En général c'est non, et on dessine dans une image que l'on affiche.

Illustration (1)


CompositeTest.bat



- Le programme dessine une ellipse rouge en alpha = 1 et, selon le choix de la règle de composition, la compose avec un rectangle bleu.
- De dessin se fait dans une `BufferedImage`, pour profiter de la couche alpha.

```
public void paintComponent(Graphics g) {
  super.paintComponent(g);
  Graphics2D g2 = (Graphics2D)g;
  if (image == null)
    image = new BufferedImage(getWidth(),getHeight(), BufferedImage.TYPE_INT_ARGB);
  Graphics2D gI = image.createGraphics();
  gI.setPaint(Color.red);
  gI.fill(ellipse);
  AlphaComposite composite = AlphaComposite.getInstance(rule, alpha);
  gI.setComposite(composite);
  gI.setPaint(Color.blue);
  gI.fill(rectangle);
  g2.drawImage(image, null, 0, 0);
}
```

Illustration (2)

- Le choix de la règle se fait par lecture de la comboBox

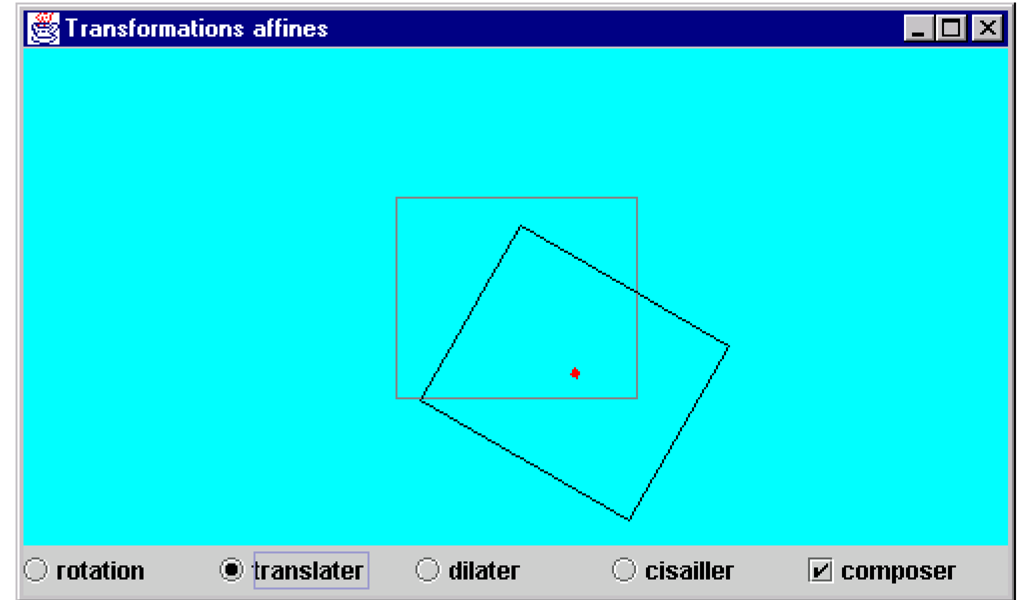
```
if (r.equals("CLEAR))
  rule = AlphaComposite.CLEAR;
else if (r.equals("SRC"))
  rule = AlphaComposite.SRC;
else if (r.equals("SRC_OVER"))
  rule = AlphaComposite.SRC_OVER;
else if (r.equals("DST_OVER"))
  rule = AlphaComposite.DST_OVER;
etc.
```

- le calcul de alpha se fait par lecture dans le curseur (et division par 100)

```
a = curseur.getValue();
alpha = (float)a / 100.0F;
```

Transformations affines

- Les transformations affines servent à modifier les coordonnées utilisateur avant affichage
- Par exemple, le repère peut être centré au milieu de la zone de dessin.
- Les transformations sont
  - *rotation*
  - *translation*
  - *dilatation*
  - *cisaillement (shear)*



- La class AffineTransform permet de créer et de composer des transformations affines. De nombreuses méthodes existent.

Opérations

- Mathématiquement, une transformation affine est représentée par une matrice 3 x 3 dont la dernière ligne est toujours (0 0 1).
- Seuls les 6 autres coefficients sont conservés. On peut donner ces coefficients explicitement, ou les faire calculer en fonction de la nature de l'opération recherchée.
- Créations:

```
AffineTransform t = new AffineTransform();
t.setToRotation(angle);
t.setToTranslation(dx, dy);
t.setToScale(sx, sy);
t.setToShear(cx,cy);
```

- Compositions:

```
t.rotate(angle);
t.translate(dx, dy);
t.scale(sx, sy);
t.shear(cx,cy);
```

Utilisation: exemple

- Quand la transformation est définie, on l'utilise en l'ajoutant à la transformation courante par

```
public void paintComponent(Graphics g) {
  super.paintComponent(g);
  Graphics2D g2 = (Graphics2D)g;
  g2.translate(getWidth() / 2, getHeight() / 2);
  g2.setPaint(Color.gray);
  g2.draw(square);
  g2.transform(t);
  g2.setPaint(Color.red);
  g2.fill(smallsquare);
  g2.setPaint(Color.black);
  g2.draw(square);
}
```

Exemple

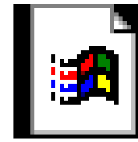
- Ici, **`composer`** est une variable booléenne qui conserve l'état de la coche.

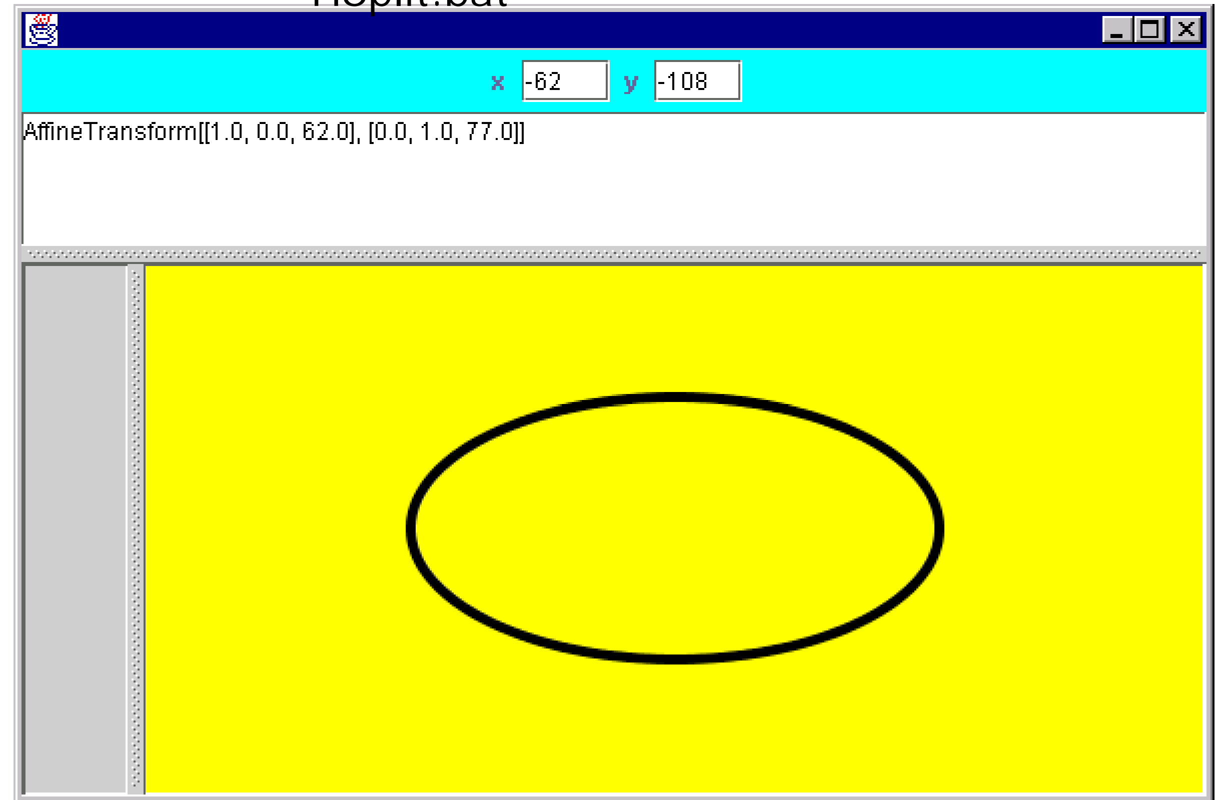
```
public void actionPerformed(ActionEvent event) {
  JToggleButton source = (JToggleButton) event.getSource(); // le bouton
  String sourceAction = source.getActionCommand(); // son libellé
  if (sourceAction.equals("composer")) {  // la coche
    composer = source.isSelected();
    return;
  }
  if (!composer) t.setToIdentity(); // composer ou non ?

  if (sourceAction.equals("rotation"))
    t.rotate(Math.toRadians(30));
  else if (sourceAction.equals("translater"))
    t.translate(20, 15);
  else if (sourceAction.equals("dilater"))
    t.scale(2.0, 1.5);
  else if (sourceAction.equals("cisailler"))
    t.shear(-0.2, 0);
  repaint();
}
```

Transformation affine implicite

- L’affichage, lors de l’exécution d’un `paintComponent`, est optimisé. Seule la zone qui doit être rafraîchie l’est vraiment, et cela dépend bien sûr de l’événement qui a provoqué l’affichage.
- Le context graphique maintient une transformation affine qui contient la translation du composant d’affichage par rapport au rectangle de réaffichage. Cette transformation *implicite* ne doit pas être ignorée, mais utilisée.
- On ajoutera donc des transformations, au lieu de les remplacer.
- Moyennant cette précaution, le décalage est transparent à l’utilisateur.

Exemple

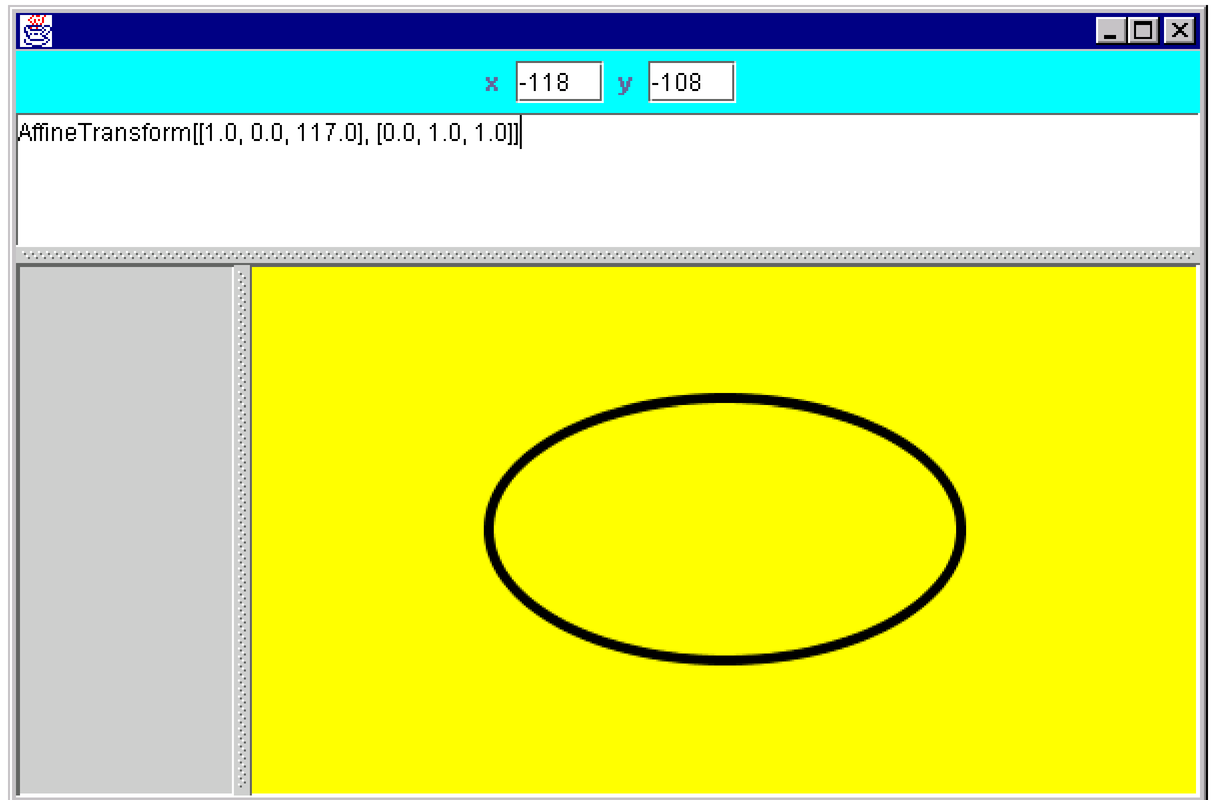




- Toute opération graphique se fait relativement au panneau jaune.
- L'origine est le coin supérieur gauche du panneau.
- Les valeurs numériques donnent la position de la souris relativement à l'origine (elle est à l'origine de la zone bleue).
- La transformation affine affichée indique la translation de l'origine du panneau jaune par rapport à l'origine de la zone qui a été redessinée. Dans le cas présent, cette origine est la zone de texte.

Exemple (suite)

- Ici, l'origine est la zone grise, après un déplacement de la barre verticale du panneau mouvant.
- Une déiconification, ou le lancement, donnent l'origine dans la zone bleue, un déplacement du panneau horizontal dans la zone blanche.



- Les informations s'obtiennent par

```
public void paintComponent(Graphics g) {
   Graphics2D g2 = (Graphics2D) g;
   ...
   AffineTransform at = g2.getTransform();
   txt.setText(at.toString());
   ...
   g2.draw(new Ellipse2D.Float(w/4,h/4, w/2, h/2));
}
```

Rendu

- Le rendu est amélioré (au dépens de la rapidité) par un ensemble de “hints” (conseils).
- Chaque conseil concerne un aspect et indique un souhait.
- Un conseil se présentent donc comme un couple: clé d’une propriété et valeur de cette propriété.

```
g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
    RenderingHints.VALUE_ANTIALIAS_ON);
g2.setRenderingHint(RenderingHints.KEY_RENDERING,
    RenderingHints.VALUE_RENDER_QUALITY);
```

- On peut aussi écrire

```
RenderingHints r = ...;
g2.setRenderingHints(r);
```

Rendu : suite

- Les aspects du rendu concernent l’anti-aliasing, la couleur, l’interpolation

```
KEY_ANTIALIASING
    VALUE_ANTIALIAS_DEFAULT
    VALUE_ANTIALIAS_OFF
    VALUE_ANTIALIAS_ON
KEY_RENDERING
    VALUE_RENDER_DEFAULT
    VALUE_RENDER_QUALITY
    VALUE_RENDER_SPEED
KEY_ALPHA_INTERPOLATION
KEY_COLOR_RENDERING
KEY_DITHERING
KEY_INTERPOLATION
    VALUE_ALPHA_INTERPOLATION_DEFAULT
    VALUE_ALPHA_INTERPOLATION_QUALITY
    VALUE_ALPHA_INTERPOLATION_SPEED
KEY_FRACTIONALMETRICS
KEY_TEXT_ANTIALIASING
```