Programmation Evènementielle

Prog. Événementielle

- Les programmes qui possèdent une IHM obéissent à un modèle de programmation appelé programmation événementielle.
- Le programme attend que se produisent des événements
- lorsque ces événements se produisent, il y réagit en déclenchant les procédures appropriées dans les objets qui sont intéressés.
- Un programme Swing est conforme à l'approche de la programmation par évènements.

Prog. Événementielle

- Java définit des événements de bas niveau fournissant une interface au système de fenêtrage de l'OS.
- Java définit aussi des événements plus sémantiques, relatifs aux objets awt et swing.
- Un thread dédié gère les événements java.
- Java définit les événements comme des objets avec une source et une cible.

Prog. Événementielle

Exemple

- si on clique la souris, c'est le système d'exploiatation qui découvre l'événement en premier.
- Il détermine quelle application contrôle la souris au moment où s'est produit l'événement,.
- il passe les informations à cette application
- Le bouton est la source de l'événement, i.e. l'objet où le clic s'est produit.
- Java crée alors un objet Event qui contient les informations associées à cet événement de clic.
- Le thread de gestion des événements appelle alors la (ou les) procédure(s) définies par le développeur associée(s) à l'événement.
- Les procédures déclenchées sont définies dans les objets cibles - et elles ont cet événement en argument.

Les écouteurs (Listeners)

- Définition:
  - ✓Un écouteur est un objet qui capte des événements se produisant sur un autre objet (ou d'un groupe d'objets, par ex. un groupe d'éléments de menu).
- un écouteur définit une méthode ayant pour argument un événement.
- Cette méthode sera déclenchée chaque fois que l'événement écouté se produira sur l'objet source.
- Tout objet, peut devenir un écouteur à condition d'implémenter une interface dite écouteur (listener).
- Il existe divers interfaces listener associées aux types d'événements.

Exemple : Un clic de bouton, produit un événement ActionEvent géré par les méthodes de l'interface ActionListener.

Procédure de gestion des évènements

- implémenter la (ou les) interface(s) listener adéquate (s)
- Redéfinir les procédures de traitement sur les événements.
- Relier la source de l'événement à la cible.

Exemple : l'écouteur doit être enregistré sur le bouton avec la méthode addActionListener ayant pour argument l'objet qui implémente l'interface listener.

Description des événements Java

- Les événements Java sont classés en deux catégories :
  - ✓ les événements de bas niveau (clavier, souris, opérations sur les fenêtres)
  - ✓ les événements sémantiques : événements spécifiques, liés à des composants swing ou awt spécifiques.
- Le package java.awt.event contient La plupart des événements et des interfaces écouteurs.
- Les événements spécifiques aux composants Swing sont dans : javax.swing.event

Événements de bas niveau

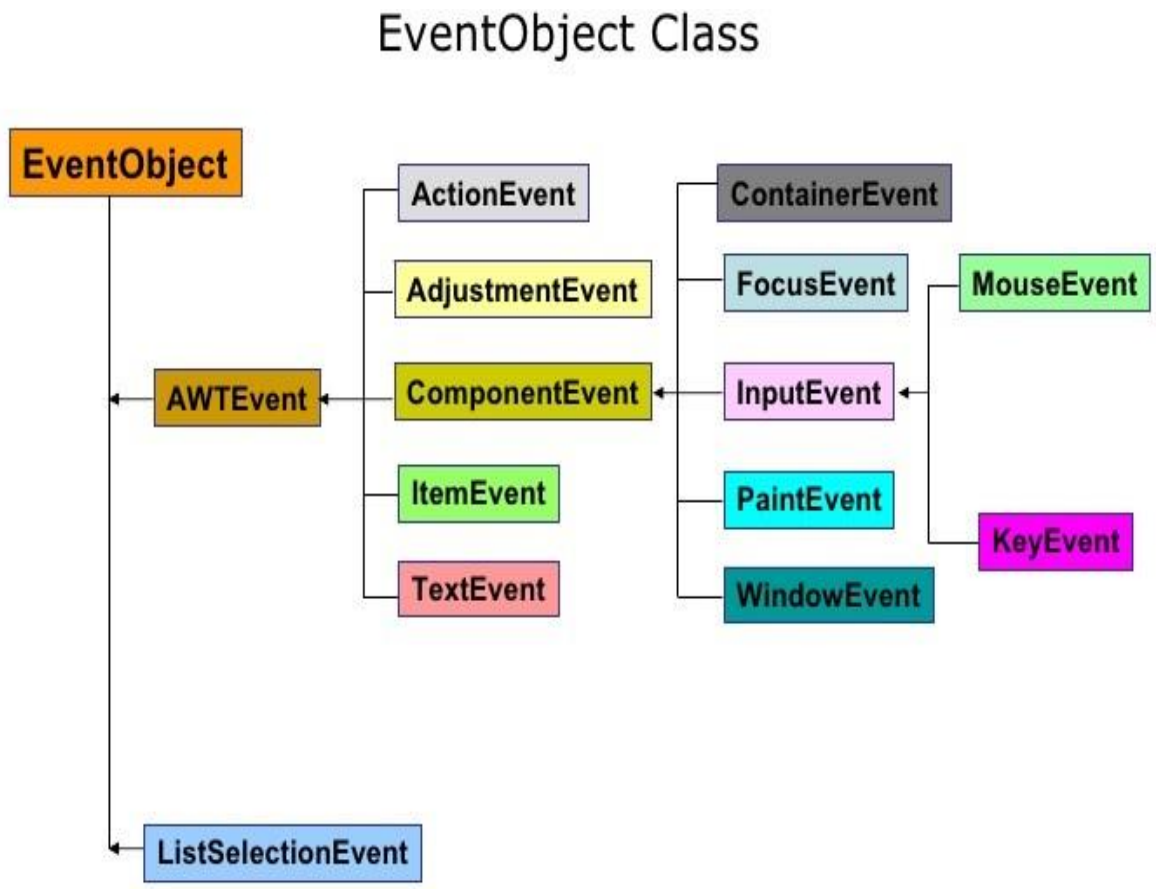
- FocusEvent : activation ou désactivation du focus du clavier sur un composant.
- MouseEvent : mouvements et clics de souris, et entrer/sortir d'un composant.
- KeyEvent : événements clavier.
- WindowEvent : dés/activation, ouverture fermeture, dés/iconification de fenêtres.
- ComponentEvent : changement de taille, position, ou visibilité d'un composant.

Événements de bas niveau (suite)

- Il existe d'autres événements de bas niveau, comme
  - ✓ PaintEvent : qui concerne la gestion du réaffichage .
  - ✓ ContainerEvent : événements associés à un conteneur (ajouts, suppressions, etc.).
- Ces événements sont traités automatiquement.

Hiérarchie des événements de bas niveau

- EventObjet contient la méthode getSource() qui renvoie l'objet source



Constantes de AWTEvent

- Ces constantes (public final) permettent d'identifier des classes d'événements.

```
1 MOUSE_EVENT_MASK
2 KEY_EVENT_MASK

3 ITEM_EVENT_MASK

4 WINDOW\_EVENT_MASK
5 MOUSE_MOTION_EVENT_MASK

6 FOCUS_EVENT_MASK
7 TEXT_EVENT_MASK

8 ADJUSTMENT_EVENT_MASK
```

les écouteurs

Il y a des interfaces écouteurs de bas niveau qui héritent de java.util.EventListener :

- WindowListener // événements fenêtre
- MouseListener // clics + entrée/sortie fenêtre
- MouseMotionListener//mouvements souris
- KeyListener //touches clavier
- FocusListener//focus clavier
- ComponentListener//configuration

Ecouteurs sur les fenêtres (WindowEvent)

```
1 windowOpened(WindowEvent e)
2 // Appelee la 1ere fois que la fenetre s'ouvre
3
4 windowClosing(WindowEvent e)
5 //Menu close du systeme
6
7 windowClosed(WindowEvent e)
8 //Appelee quand on a ferme la fenetre
9
10windowActivated(WindowEvent e)
11//Quand la fenetre est activee (ex:clic dessus)
12
13windowDeactivated(WindowEvent e)
14//Quand la fenetre est desactivee
15
16windowIconified(WindowEvent e)
17//Quand la fenetre est reduite a une icone
18
19windowDeiconified(WindowEvent e)
20//Quand on restore la fenetre d'une icone
```

MouseListener

L'interface MouseListener offre les méthodes de gestion suivantes :

```
1
2 mouseClicked(MouseEvent e)
3 //Enfoncer et relacher: clic sur un composant
4
5 mousePressed(MouseEvent e)
6 //Enfoncer sur un composant
7
8 mouseReleased(MouseEvent e)
9 //Relacher sur un composant
10
11mouseEntered(MouseEvent e)
12//Entrer dans la zone d'un composant
13
14mouseExited(MouseEvent e)
15//Quitter la zone d'un composant
```

MouseMotionListener

L'interface MouseMotionListener offre les méthodes de gestion suivantes :

```
1mouseMoved(MouseEvent e)
2//Mouvement de la souris
3
4mouseDragged(MouseEvent e)
5//Drag : mouvement + bouton enfonce
```

MouseInputListener

- Elle implémente à la fois un MouseListener et un MouseMotionListener.
- Il n'existe pas de méthode addMouseInputListener, et quand on l'utilise.
- Il faut enregistrer l'écouteur deux fois : une fois avec addMouseListener et une autre fois avec addMouseMotionListener.

KeyListener

L'interface KeyListener offre les méthodes de gestion suivantes :

```
1 keyTyped(keyEvent e)
2 //Cle pressee et relachee
3 keyPressed(keyEvent e)
4 //Cle pressee
5 keyReleased(keyEvent e)
6 //Cle relachee
```

FocusListener

Méthodes de l'interface FocusListener :

```
1 focusGained(FocusEvent e)
2 //Quand un composant prend le focus clavier

3 focusLost(FocusEvent e)
4 //Quand il le perd
```

ComponentListener

Méthodes de l'interface ComponentListener :

```
1componentHidden(ComponentEvent e)
2//Le composant est rendu invisible
3componentMoved(ComponentEvent e)
4//La position du composant change
5componentResized(ComponentEvent e)
6//La taille du composant change
7componentShown(ComponentEvent e)
8//Le composant est rendu visible
```

Exemple
Lines 1–21 / 30

```
1 package ma.ensah;
2
3 import java.awt.event.WindowEvent;
4 import java.awt.event.WindowListener;
5
6 import javax.swing.JFrame;
7
8 public class Dessinateur extends JFrame implements WindowListener {
9
10public static void main (String[] args) {
11    Dessinateur theApp = new Dessinateur();
12 }
13public Dessinateur() {
14    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15    addWindowListener(this);
16    setSize(400, 400);
17    setVisible(true);
18 }
19
20// Methode pour l'evenement windowClosing
21public void windowClosing(WindowEvent e ) {
```

Exemple
Lines 20–40 / 30

```
1  // Methode pour l'evenement windowClosing
2  public void windowClosing(WindowEvent e ) {
3        System.out.println("windowClosing");
4        System.exit(0); // termine l'application
5  }
6  // fonctions de l'interface listener que nous devons implementer
7  public void windowOpened(WindowEvent e ) ↙
       ↳ {System.out.println("windowOpened");};
8  public void windowClosed(WindowEvent e ) ↙
       ↳ {System.out.println("windowClosed");};
9  public void windowIconified(WindowEvent e ) ↙
       ↳ {System.out.println("windowIconified");};
10 public void windowDeiconified(WindowEvent e ) ↙
       ↳ {System.out.println("windowDeiconified");};
11 public void windowActivated(WindowEvent e ) ↙
       ↳ {System.out.println("windowActivated");};
12 public void windowDeactivated(WindowEvent e ) ↙
       ↳ {System.out.println("windowDeactivated");};
13 }
```

Les écouteurs/Limitations

- il faut implémenter toutes les méthodes de l'interface correspondante, y compris celles dont on ne se sert pas.
- Cet inconvénient peut être supprimé grâce aux classes Adapter.

Classes Adapter

- **Définition** : Ce sont des classes qui implémentent une interface (ici écouteur) mais dont les méthodes n'ont pas de code : elles ne font rien.
- On peut alors étendre la classe Adapter choisie en redéfinissant uniquement les méthodes dont on a besoin.

Classes Adapter

- Il y a une classe Adapter pour chacun des écouteurs de bas niveau définis dans le package java.awt.event
- Il y a une classe définie dans le package javax.swing.event étendant l'interface MouseInputListener.
  - ✓ FocusAdapter
  - ✓ MouseAdapter
  - ✓ WindowAdapter
  - ✓ KeyAdapter
  - ✓ MouseMotionAdapter
  - ✓ MouseInputAdapter

Classes Adapter
Etendre directement une classe Adapter

```
1 public class MaClass extends MouseAdapter {
2   ...
3   unObject.addMouseListener(this);
4   ...
5   public void mouseClicked(MouseEvent e) {
6     ...
7     // l'implementation de la methode
8     // associee a l'evenement vient ici ...
9   }
10 }
```

Classes Adapter
Passer par une classe interne

```
1window.addWindowListener(new WindowHandler());
2// classe interne WindowHandler
3
4   //pour les evenements de fermeture
5   class WindowHandler extends WindowAdapter {
6      // Methode pour WINDOW_CLOSING event
7      public void windowClosing( WindowEvent e ) {
8         window.dispose();
9         System.exit(0);
10     }
11
```

Événements sémantiques

Ce sont des événements liés à des opérations spécifiques sur certains composants : sélection d'éléments de menu, clic sur un bouton, etc.

- Il y a trois classes de base d'événements sémantiques dérivés de AWTEvent.
- Il existe des événements sémantiques spécifiques aux composants Swing dérivés également de cette classe.

Événements sémantiques

- EventObject
  - ✓ AWTEvent
    - ActionEvent
    - ItemEvent
    - AdjustmentEvent
- Ces trois événements sémantiques sont dans java.awt.event.

ActionEvent

- Déclenché par une action sur un composant réactif : clic sur un item de menu ou sur un bouton.
- Émis par les objets :
  - ✓ JButton, JToggleButton, JCheckBox
  - ✓ JMenu, JMenuItem, JCheckBoxMenuItem,JRadioBoxMenuItem.
  - ✓ JTextField

AdjustmentEvent

- il se produit quand un élément ajustable, comme une JScrollbar, est ajusté.
- Émis par JScrollbar.

Écouteurs pour événements sémantiques

interface	listener
ActionListener	void actionPerformed(ActionEvent e)
ItemListener	void itemStateChanged(ItemEvent e)
AdjustmentListener	void adjustmentValueChanged(AdjustmentEvent e)

Exemple

```
…
Button b = new Button(" bouton ");
…
void public actionPerformed(actionEvent evt) {
  Object source = evt.getSource();

  if (source == b) // action a effectuer
}
```

Evénements sémantiques

- D'autres évènements sont supportés :
  - ✓ ChangeEvent quand on modifie l'état d'un bouton.
  - ✓ Les dérivés de JMenuItem génèrent des MenuDragMouseEvent et des MenuKeyEvent.
  - ✓ Une JList génère des SelectionEvent.
  - ✓ Les modèles associés aux listes et aux tables génèrent des ListDataEvent et des TableModelEvent (envoyés aux vues quand des changements se produisent sur le modèle).

Le contexte graphique

- En Swing, Pour dessiner il faut utiliser un contexte graphique (instance Graphics)
- Un objet Graphics permet de :
  - dessiner dans le composant concerné (JComponent ou Image)
  - définir la couleur courante : set/getColor()
  - définir la fonte courante : set/getFont()
  - gérer une translation : translate(int x,int y)
  - . . .

Les primitives de dessin

- Les méthodes suivantes utilise l'état courant :

```
1   drawArc(int x, int y, int width, int height, intstartAngle, int arcAngle)
2   drawImage(Image img, int x, int y, ImageObserverobserver)
3   drawLine(int x1, int y1, int x2, int y2)
4   drawOval(int x, int y, int width, int height)
5   drawPolygon(int[] xPoints, int[] yPoints, intnPoints)
6   drawRect(int x, int y, int width, int height)
7   drawRoundRect(int x, int y, int width, int height, int arcWidth, int ↙
        ↳ arcHeight)
8   drawString(String str, int x, int y)
9   fillArc(int x, int y, int width, int height, intstartAngle, int arcAngle)
10  fillOval(int x, int y, int width, int height)
11  fillPolygon(int[] xPoints, int[] yPoints, intnPoints)
12  fillRect(int x, int y, int width, int height)
13  fillRoundRect(int x, int y, int width, int height, int arcWidth, int ↙
        ↳ arcHeight)
14  ...
```

Obtention/Libération d'un contexte graphique

- Implicitement :
  - ✓ Récupérer le Graphics passé en argument à paintComponent (méthode de JComponent)
- Explicitement :
  - ✓ Récupérer le Graphics d'un composant ou d'une image avec getGraphics()
  - ✓ Cloner un Graphics existant avec createGraphics()
- La méthode dispose() permet de libérer le contexte obtenu pour ne pas monopoliser les ressources système.

Dessiner : Premier exemple

```
1  public class Pacman extends JComponent {
2    @Override protected void paintComponent(Graphics g) {
3  super.paintComponent(g);
4  if (isOpaque()) {
5          g.setColor(getBackground());
6          g.fillRect(0,0,getWidth(),getHeight());
7  }
8  g.setColor(Color.YELLOW);
9  g.fillArc(0,0,getWidth(),getHeight(),35,280);
10 g.setColor(Color.BLACK);
11 g.fillOval((int)(getWidth()*.65),(int)(getHeight()*0.15), ↙
       ↳ (int)(getWidth()*.08),(int)(getHeight()*0.08));
12   }
13 }
```

Attributs du Contexte Graphique

- Sept attributs sont définis dans un contexte graphique de type Graphics2D
- Certains servent pour tout : Composite, Transform, Hint, Clip
- Certains ne servent que pour les formes : Paint, Stroke
- Un attribut ne sert qu'au texte : Font

Attributs du Contexte Graphique : Composite

- Détermine comment l'objet à dessiner doit se mélanger à ce qui est déjà dessiné.
- Utile pour rendre les objets transparents, faire des mélanges de couleurs.

```
1  Graphics2D g2D = (Graphics2D) g;
2  rectangle = ...;
3  ellipse = ...;
4  g2D.setPaint(new Color(102,0,204));
5  g2D.translate(70,100);
6  g2D.fill(rectangle);
7
8  g2D.setComposite(AlphaComposite.getIns ↙
       ↳ tance(AlphaComposite.SRC_OVER,0.5f));
9  g2D.setPaint(new Color (255,51,204));
10 g2D.translate(70,60); g2D.fill(ellipse);
11
12 g2D.setPaint(new Color (255,204,102));
13 g2D.setStroke(new BasicStroke(4));
14 g2D.draw(ellipse);
```

Attributs du Contexte Graphique : Composite

- Prohiber le mélange des couleurs :

```
1 g2D.setPaintMode();
2 gd2D.setComposite(new AlphaComposite.SrcOver)
```

- Autoriser la mélange des couleurs : Soit deux pixels $A(RA,GA,BA)$ et $B(RB,GB,BB)$
- Le rsultat est un pixel $P(RP,GP,BP)$ où : $XP = XA \times CA + XB \times CB$ avec CA et CB des poids définis.
- AlphaComposite contient huit modèles de composition : combinaisons CA, CB.

Attributs du Contexte Graphique : Composite/Transparence

- Utiliser le modèle de composition :

  AlphaComposite.SRC_OVER avec : $CB = 1 - AB, CA = 1$;

```
1   g2D.setComposite(AlphaComposite.getInstance(
        ↳ AlphaComposite.SRC_OVER,alpha));
```

- Plus alpha est petit plus la transparence est grande

Attributs du Contexte Graphique : Paint

- Trois modèles de remplissage : Color, GradientPaint, TexturePaint
- Un dégradé défini par 5 paramètres :
  - ✓ Deux points : D1, D2
  - ✓ Deux couleurs : C1 et C2
  - ✓ Une stratégie de parcours (optionel)
- Class GradientPaint qui encapsule ces informations.

Rendu

- La méthode Paint() appelle par défaut
  1. paintComponent()
  2. paintBorder()
  3. paintChildren()
- Pour redéfinir le rendu du composant Swing il faut redéfinir

  paintComponent()
- Pour appeler un rafraîchissement à la demande :
  - ✓repaint()
  - ✓repaint(Rectangle r) ne rafraîchit que la zone couverte par le rectangle

Contrat d’opacité

Si le composant crée ne remplit pas toute la zone de dessin, il faut

obéir au contrat d’opacité :

- Si le composant est opaque, on remplit la zone avec la couleur de fond
- Sinon, on ne fait rien

```
1  if (isOpaque()) {
2          g.setColor(getBackground());
3          g.fillRect(0,0,getWidth(),getHeight());
4  }
```

Exemple

```
1  public class VolatileScratch extends JComponent implements
       ↳ MouseMotionListenr {
2  public VolatileScratch(int width,int height) {
3  setPreferredSize(new Dimension(width,height));
4  addMouseMotionListener(this);
5
6  public void mouseDragged(MouseEvent e) {
7  Graphics g=getGraphics();
8  try {
9          g.setColor(Color.GREEN);
10         g.fillOval(e.getX()-7,e.getY()-7,14,14);
11 } finally {
12         g.dispose();
13 }
14 }
15
16 }
17 ...
18 }
```

Problème et Solution

- Quand on doit rafraîchir la fenêtre, on perd ce qui n'était plus visible !
- Solution : dessiner dans une image

Exemple

```
1  public class PersistentScratch extends JComponent implements
       ↳ MouseMotionListener{
2  private final BufferedImage image;
3  public PersistentScratch(int width,int height) {
4  setPreferredSize(new Dimension(width,height));
5  final BufferedImage image=new
       ↳ BufferedImage(width,height,BufferedImage.TYPE_INT_ARGB);
6  this.image=image;
7  addMouseMotionListener(this);
8  @Override public void mouseDragged(MouseEvent e) {
9  Graphics g=image.getGraphics();
10 try {
11 g.setColor(Color.GREEN);
12 g.fillOval(e.getX()-7,e.getY()-7,14,14);
13 paintImmediately(0,0,getWidth(),getHeight());
14 } finally {
15 g.dispose();
16 }
17 }
18 }
19 }
20 @Override protected void paintComponent(Graphics g) {
21 g.drawImage(image,0,0,null);
```

ImageObserver

- Le dernier paramètre de drawImage est un ImageObserver, qui sert à être prévenu du chargement de l'image.
- Dans notre cas, null signifie que l'on ne gère pas la notification de chargement.

```
1  @Override protected void paintComponent(Graphics g) {
2          g.drawImage(image,0,0,null);
3  }
```

Le clipping

- Zone de clipping=pochoir appliqué au dessin qui va suivre
- Exemple : utilisation d'un disque pour n'afficher qu'une portion d'image
  - setClip(int x,int y,int width,intheight)
  - setClip(Shape clip)

```
1   protected void paintComponent(Graphics g) {
2   g.setColor(getBackground());
3   g.fillRect(0,0,getWidth(),getHeight());
4   if (x==-1 && y==-1) {
5   return;
6   }
7   Shape clip=new Ellipse2D.Float(x-60,y-60,120,120);
8   g.setClip(clip);
9   g.drawImage(image,0,0,null);
10  }
```

Graphics2D

Graphics2D étend Graphics et permet de gérer plus de paramètres :

- Le pinceau à utiliser
- La transparence
- Le dessin de formes complexes
- Les options de rendus
- Les transformations affines

En Swing, on peut toujours caster un Graphics en Graphics2D (type effectif des paramètres instance de Graphics)

Le pinceau

- Interface : Stroke
- Implémentation : BasicStroke

Paramètres supportés :

- Epaisseur du trait
- Forme des extrémités de segment
- Forme des angles d'un polygone
- Dessin en pointillé

La transparence

- On peut définir la transparence du dessin par deux approches :
- Utiliser une couleur translucide obtenu par

```
1        new Color(int r,int g,int b,int a)
```

- Définir un Composite, obtenu avec :

```
1        AlphaComposite.getInstance(int rule,float alpha)
```

  - ✓ rule mode de combinaison entre le dessin et l'arrière-plan
  - ✓ 0<alpha<1 : 0.0=transparent, 1.0=opaque

Exemple :

```
1  public class GhostButton extends JButton {
2  private Composite ↙
       ↳ composite=AlphaComposite.getInstance(AlphaComposite.SRC_OVER,0.5f);
3  public GhostButton(String text) {
4  super(text);
5  /* We don't want the background of the button to be painted */
6  setOpaque(false);
7  }
8  @Override protected void paintComponent(Graphics g) {
9  Graphics2D g2=(Graphics2D)g;
10 Composite old=g2.getComposite();
11 g2.setComposite(composite);
12 super.paintComponent(g);
13 g2.setComposite(old);
14 }
15 ...
16 }
```

La transparence Exemple

un JLabel dont la couleur est transparente :

```
1   JLabel label=new JLabel("I'm a blue+composite label");
2   label.setForeground(new Color(0,100,255,150));
```

Interface Shape

- Sert à définir des formes.
  - ✓Formes prédéfinies :
  - ✓Ellipse2D
  - ✓Rectangle2D
  - ✓RoundRectangle2D
  - ✓Line2D
  - ✓CubicCurve2D, QuadCurve2D
  - ✓…

pour les formes XXX2D, on a 2 implémentations qui diffèrent par leur précision :

- XXX2D.Float
- XXX2D.Double

Shape

- Pour dessiner le contour d'une Shape, en utilisant le pinceau
- courant et la couleur courante :

  g.draw(shape);
- Pour la remplir avec la couleur courante :

  g.fill(shape);

Texte : Rendu simple ou avancé

- Rendu simple (hérité de Graphics)

```
1  g2D.setFont(new Font(...));
2  g2D.drawString("Rendu simple", 40,50);
```

- Rendu plus élaboré

```
1  TextLayout, LineBreakMeasurer
```

- Rendu très élaboré

```
1  drawGlyphVector(GlyphVector g, float x, float y)
```