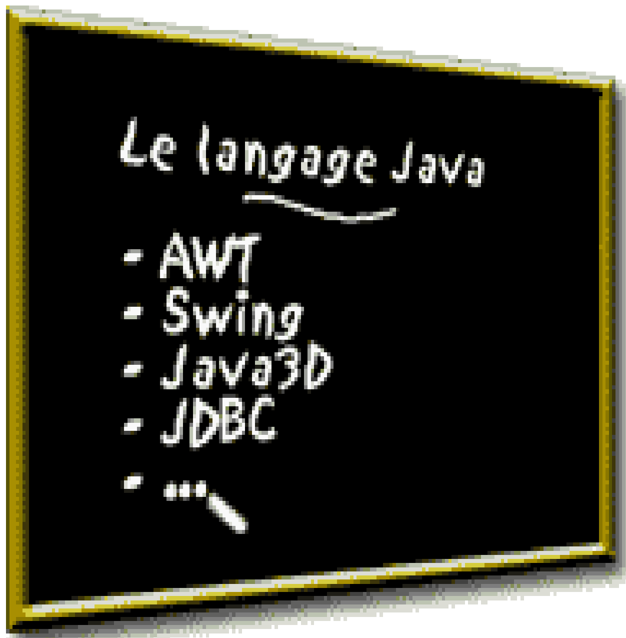
Cours 5 : Le dessin



- Comment les composants s'affichent ?
- La méthode paint
- La méthode paintComponent
- La méthode repaint
- La classe Graphics
- La classe Graphics2D

Affichage déclenché par le système ou l'application

- Le système demande à un composant de se *peindre* quand :
  - le composant est rendu visible pour la première fois sur l 'écran
  - le composant a été redimensionné
  - le rendu du composant a été endommagé (masqué puis à nouveau visible, transformé en icône ou l'inverse, activé…)

  Exemple : show adressé à une JFrame provoque un affichage
- Affichage déclenché par l'application :
  - Le composant doit se repeindre pour refléter un changement dans son état interne

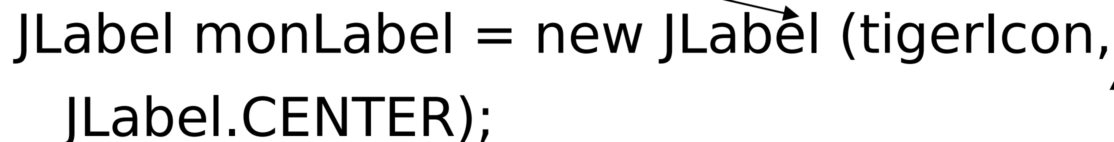
Jusque là, on n'a pas eu besoin de s'intéresser à l'affichage graphique. Pourquoi ?

- Les composants Swing gèrent seuls leurs affichage et savent se repeindre quand nécessaire
  - Quand ils sont inclus dans un composant qui doit lui-même se réafficher
  - Quand leur état interne est modifié
    - exemple : méthode `setText` adressée à un `JTextComponent` est évoquée

Les besoins graphiques « primaires » de l 'application peuvent être résolus en utilisant des composants

- Si l'application a besoin d'afficher du texte
  - Elle peut se servir d'un JLabel et de sa méthode `setText`
- Si l'application a besoin d'afficher une image
  - Elle peut l'afficher dans un JLabel (ou autre):

```
Icon tigerIcon = new ImageIcon("SmallTiger.gif");
```



```
JLabel monLabel = new JLabel (tigerIcon,
  JLabel.CENTER);
```

Principe d ’affichage d'un composant

- La demande de *repeinte* du composant (qu’elle soit déclenchée par le système ou par l’application) est demandée par l’intermédiaire de la méthode

  public void paint (Graphics g) .

- Même méthode quel que soit le composant

- Quand cette méthode est appelée, le paramètre, instance de Graphics (contexte graphique) est pré-configuré par le système avec un état approprié pour dessiner sur ce composant particulier

Et s'il y a des composants imbriqués ?

- Si la fenêtre principale contient d'autres composants, l'affichage suit l'ordre d'imbrication des composants

Imbrication des composants

JFrame

→ zone de travail

→ JButton

→ JLabel

```
c= this.getContentPane();
c.add(newJButton(“I'm a Swing button!”);
c.add(new JLabel(“Number of buttonClicks:0”));
```

Décomposition de l'affichage graphique de l ’exemple

1. La JFrame se peint puis demande à son aire de travail de se peindre
2. Le content pane se peint (en gris) et demande à chacun de ses composants imbriqués de se peindre
3. Le bouton se peint ~ peint son background puis peint son texte
4. Le label se peint ~ peint son texte

Remarque : c'est là que l'opacité du composant peut jouer

Décomposition de l'affichage graphique en général

- Un composant (instance de JComponent) se peint lui-même d’abord puis peint les composants qu'il contient (ses enfants)

- Ordre :
  - 1. Se peindre lui-même
  - 2. Peindre son bord
  - 3. Peindre ses enfants (s'il en a)

Techniquement, comment ça se passe ?

- **La méthode d'affichage graphique d'un composant est toujours la même : public void paint(Graphics g)**

- Sa signature est **fixe**, le paramètre est fourni par le système

- Cette méthode appelle dans l'ordre les méthodes en leur passant son propre paramètre g :

  - **1.** Se peindre soi-même : **void paintComponent(Graphics g)**
  - **2.** Peindre son bord : **void paintBorder(Graphics g)**
  - **3.** Peindre ses enfants : **void paintChildren(Graphics g)**

paintChildren :

Que faire pour faire du "Customer painting" ? (affichage graphique spécifique)

- **On ne redéfinit ni paint, ni paintBorder, ni paintChildren mais seulement void paintComponent (Graphics g)**
- **paintComponent** est une méthode de **JComponent**
- donc impossible de faire de redéfinir cette méthode pour la zone de travail d'une JFrame ou JApplet

Solution : Définir une sous-classe (appropriée) de **JPanel** correspondant à la zone où on veut faire de l'affichage graphique. Et rédéfinir dans cette classe la méthode

void paintComponent (Graphics g)

Comment faire du "Customer painting" ? (affichage graphique spécifique)

- En swing, on ne redéfinit ni paint, ni paintBorder, ni paintChildren mais seulement paintComponent
- Définir une classe spécifique (héritant de JComponent ou d 'une de ses sous-classes) correspondant au composant dans lequel on veut faire de l'affichage graphique
- Rédéfinir dans cette classe la méthode

void paintComponent (Graphics g)

où on met le code indiquant ce qu'il faut peindre dans le composant après avoir fait un appel à **super.paintComponent(g)**.

La classe Graphics (contexte graphique)

- Graphics = Boîte à outils de dessin
- la classe Graphics :
  - Encapsule les caractéristiques courantes : couleur du fond, couleur du trait, style de trait, police,…
  - Offre toutes les méthodes pour dessiner, colorier,…
- A tout composant est associé une instance de la classe Graphics qui est pré-configurée par le système avec un état approprié pour dessiner sur le composant

Une instance de Graphics pour un composant

- L'instance de Graphics, `g`, associée à un composant est préconfigurée avec un état approprié pour dessiner sur ce composant :
  - l'objet couleur de `g` prend pour valeur celle du foreground du composant
  - sa fonte est celle de la fonte du composant (instance de la classe `Font`)
  - sa translation est fixée de sorte que les coordonnées (0,0) représente le coin supérieur gauche du composant
  - son rectangle est fixé à la zone qui doit être repeinte (rectangle de découpe ou ***clip rectangle***)

La classe Graphics fournit des méthodes pour changer son état

- Translater la zone d'affichage

  translate(x,y)
- Changer la zone de clipping (forcément + petite)

  get/setClip(x,y,width,height)
- Changer la couleur courante

  get/setColor(color)
- Changer la fonte courante

  get/setFont(font)

La classe **Graphics** fournit des méthodes de dessin (qui utilisent l'état courant)

- Afficher une ligne, rectangle, ellipse ou polygone

  drawLine(x1,y1,x2,y2)

  drawRect(x,y,width,height)

  drawOval(x,y,width,height)

  drawPolygon(polygon)

Les coordonnées ont pour origine le sommet supérieur gauche du rectangle graphique

- Afficher remplit un rectangle, ellipse ou polygone

  fillRect(x,y,width,height)

  fillOval(x,y,width,height)

  fillPolygon(polygon)

Premier exemple : diagonale de la fenêtre toujours tracée

Démo

```
public class FenAfficheDiagonale extends
JFrame          {

public FenAfficheDiagonale(String titre, int w,
int h)  {super(titre);
        this.initialise();
        this.setSize(w,h);
        this.show();
}

public void initialise()        {
        Container c= this.getContentPane();
        PanelDiagonale pan=new
PanelDiagonale();
        c.add(pan, «  Center  »);
```

La nouvelle classe pour le graphique

```
public class PanelDiagonale extends
JPanel {

  public void
paintComponent(Graphics g) {
        super.paintComponent(g);
        int larg= this.getWidth();
        int haut= this.getHeight();
        g.drawLine(0, 0, larg, haut);
        }
 }
```

Les dimensions sont calculées <u>dynamiquement</u> à chaque nouvel appel

Fonctionnement de l'affichage de la fenêtre

- FenAfficheDiagonale contient un composant, instance de PanelDiagonale, "qui prend toute la place"
- Quand FenAfficheDiagonale s'ouvre, se redimensionne,...
  - la fenêtre se peint elle-même
    - puis peint son seul fils (un PanelDiagonale) en appelant la méthode paintComponent définie dans PanelDiagonale
- Mais on ne sait pas <u>quand</u> l'appel a lieu

Attention : on ne maîtrise pas quand la méthode **paintComponent** est appelée

- C'est le composant qui doit connaître, à tout instant, les informations sur l'affichage qu'il doit réaliser

==> Solution : le composant doit avoir ces informations en variables membres

- Ces informations doivent être toujours "à jour", de façon à refléter, à tout instant, ce qui doit être affiché dans le composant

- Elles seront exploitées par la méthode **paintComponent**

Continuation de l'exemple : choisir, dans un menu, la couleur d'affichage de la diagonale

Démo

- Analyse : qu'est-ce qui change ?
  - Au niveau graphique : Munir la fenêtre d'un menu
  - Au niveau événementiel : Créer une inner-classe `CouleurListener` écoutant les événements du menu, implémentant `ActionListener`
  - Que doit faire la méthode `actionPerformed` ?
    - Récupérer la couleur sélectionnée
    - Modifier la couleur d 'affichage du panneau graphique
  - Et surtout, le panneau graphique doit toujours s'afficher dans la couleur choisie.

Analyse (suite)

- Le panneau graphique doit connaître à tout moment la couleur dans laquelle il va afficher le rectangle
  - ⇒ mettre une variable membre `Color coul` dans `PanelDiagonale`
  - ⇒ `coul` sera utilisé dans la méthode `paintComponent`
- Il doit y avoir une communication entre l 'objet fenêtre et l 'objet panneau graphique pour que l'écouteur `CouleurListener` puisse changer cette couleur :
  - ⇒ Mettre le panneau graphique en variable membre de la fenêtre
  - ⇒ Mettre une méthode `setColor` dans `PanelDiagonale`
  - ⇒ Adresser cette méthode au panneau graphique
- L'écouteur `CouleurListener` doit pouvoir forcer le réaffichage du panneau graphique

Mais attention ! Le programmeur ne peut pas faire appel directement à la méthode paintComponent

- L'affichage dans un composant n'est jamais fait directement mais de manière asynchrone
- S'il veut modifier l'affichage :
  - **Il doit utiliser la méthode repaint()**
- **repaint()** adressée à un composant déclenche l'exécution de sa méthode **paintComponent** et donc le réaffichage du composant

La fenêtre

```
public class FenDiagonaleMenu extends JFrame {
  private PanelDiagonale pan;

public FenDiagonaleMenu(String titre, int w, int h)   {

        super(titre);
        this.initialise();
        this.setSize(w,h);
        this.show();
        }

public void initialise()       {
        pan=new PanelDiagonale();
        this.getContentPane().add(pan, «  Center  »);
        this.initialiseMenu();
        }
```

```
public void initialiseMenu()  {
    JMenuBar jmb = new JMenuBar();
    this.setJMenuBar(jmb);

    JMenu mdef = new JMenu ("Couleur");
    jmb.add(mdef);

    JMenuItem iBleu= new JMenuItem ("Bleu");
    mdef.add(iBleu);
    JMenuItem iRouge= new JMenuItem ("Rouge");
    mdef.add(iRouge);

     iRouge.addActionListener(new CouleurListener());
     iBleu.addActionListener(new CouleurListener());

     }
```

L 'inner-classe : écouteur

```
class CouleurListener implements ActionListener  {

public void actionPerformed(ActionEvent e) {

        String s=e.getActionCommand();

        if (s.equals("Rouge"))  pan.setColor(Color.red);
                else  pan.setColor(Color.blue);

        pan.repaint();
        }
}// fin CouleurListener
}// fin FenDiagonaleMenu
```

```
public class PanelDiagonale extends JPanel {
 private Color coul;

public PanelDiagonale ()  {
        coul=Color.black;
        }

public void setColor(Color c)      {
        coul=c;
}

public void paintComponent(Graphics g) {
        super. paintComponent(g);
        g.setColor(coul);
        int larg= this.getWidth();
        int haut= this.getHeight();
        g.drawLine(0, 0, larg, haut);
   }
}
```

Spécificités de Swing

- Par défaut, l'affichage graphique se fait par double-bufferisation
  - L'affichage est exécuté dans un buffer en mémoire (off-screen buffer) puis une fois l'affichage totalement effectué sur ce buffer, ce buffer est vidé (flushed) à l'écran.
- Intérêts :
  - meilleure performance
    - Exemple : si un composant est opaque, rien ne sera peint derrière lui en final.
  - pas de clignotement

Supplément sur la classe Graphics
Afficher une image

g.drawImage(Image, x, y, ImageObserver)

- **ImageObserver** est une interface
- L'argument de type **ImageObserver** passé aux méthodes est souvent this (doit être un objet d'une classe qui implémente l'interface ImageObserver, par exemple Component).

Graphics : Afficher une image
Première étape : Créer une instance de la classe Image partir d 'un fichier

- Lecture dans un fichier local

```
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);
```

Deuxième étape : utiliser la méthode drawImage de la classe Graphics

```
public class PanelImage expends JPanel          {
private Image image;

public PanelImage(String nom)   {
        image=.....   // définie à partir du nom


public void paintComponent(Graphics g) {
        super. paintComponent(g);
          g.drawImage(image, 0, 0, this);
}
```

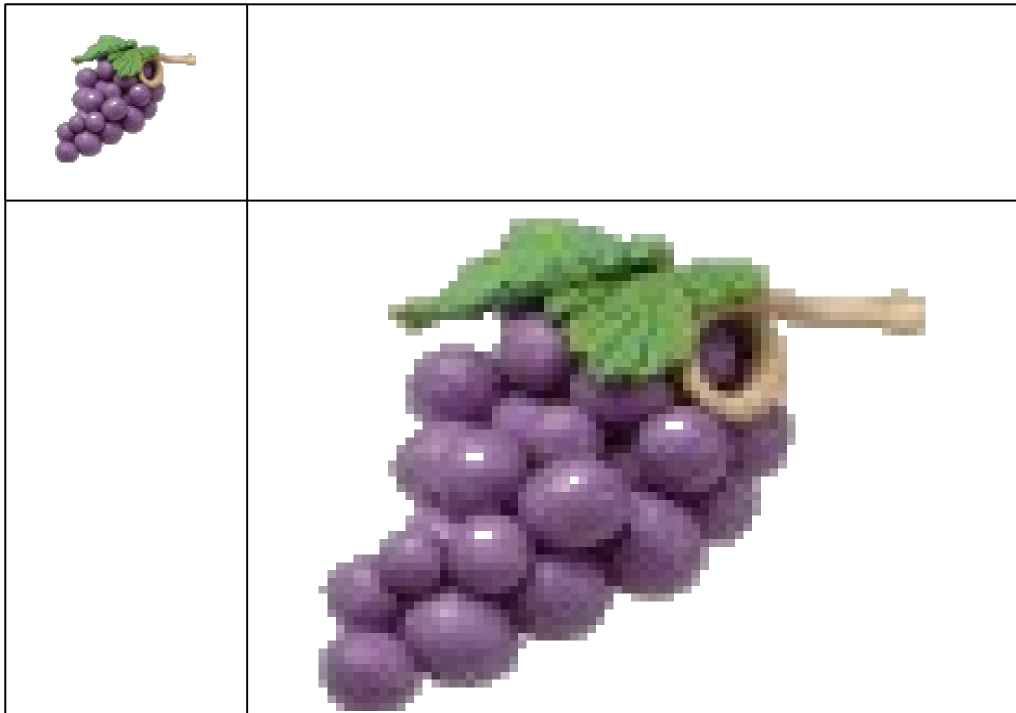
Affichage de l'image avec déformation éventuelle

drawImage(Image img, <u>int x, int y, int width, int height,</u>
ImageObserver observer)

- affiche l'image adaptée au rectangle indiqué
- l'image est mise à la bonne échelle

Exemple : Affichage de l'image en double
1 fois en haut à gauche dans son format normal
1 autre fois à l'échelle du reste

Démo



La classe graphique

```
class PanelImageEchelle extends JPanel {
  private Image im;

  public PanelImageTriple(String nomImage)  {
      im = Toolkit.getDefaultToolkit().getImage(nomImage);
     this.setBackground(Color.white);
  }

  public void paintComponent(Graphics g){
    super.paintComponent(g);
    int hImage= im.getHeight(this);
    int lImage = im.getWidth(this);
    int hPanel= this.getHeight();
    int lPanel = this.getWidth();

     g.drawImage(im, 0, 0, this); //85x62 image

     g.drawImage(im, lImage, hImage, lPanel-lImage, hPanel-
hImage, this);
```

Pour des dessins plus sophistiqués : l’ API JAVA2D

- capacités graphiques avancées en deux dimensions
- graphismes, images, et textes 2D
- extension java.awt et utilisation de nouvelles classes
- Caractéristiques Java2D
  - structure de remplissage (dégradés, motifs)
  - personnaliser la largeur et le style d ’un trait

la Classe Graphics2D

- La classe **Graphics2D** hérite de la classe **Graphics**
- Depuis java 1.2, l 'objet **Graphics** fourni à la méthode paint est en réalité une instance de la classe **Graphics2D**
- Pour maintenir la compatibilité ascendante avec java 1.1, la signature de la méthode

  void paint(Graphics g) est inchangée

Se servir de la classe Graphics2D

- Pour utiliser Java2D, il faut caster le paramètre (déclaré instance de Graphics) en un objet de la classe Graphics2D :

```
public void paintComponent (Graphics g) {
    super.paintComponent(g);
    Graphics2D g2D= (Graphics2D) g ;
    g2D....
```

Exemples basiques de code

- Utiliser des interfaces/classes de java.awt.* pour
  - Choisir l'outil de dessin du trait (pour les contours)

```
Stroke stroke = new BasicStroke(5);
g.setStroke(stroke);
```

  - choisir l'outil de remplissage (couleur : Color, dégradé, motif)

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

  - définir la forme de découpage

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