Dessiner

- Utiliser la classe java.awt.Graphics qui contient les méthodes de dessin.
- La classe Graphics doit être importé pour être connue.
- Dans une applette il suffit de redéfinir la méthode paint()

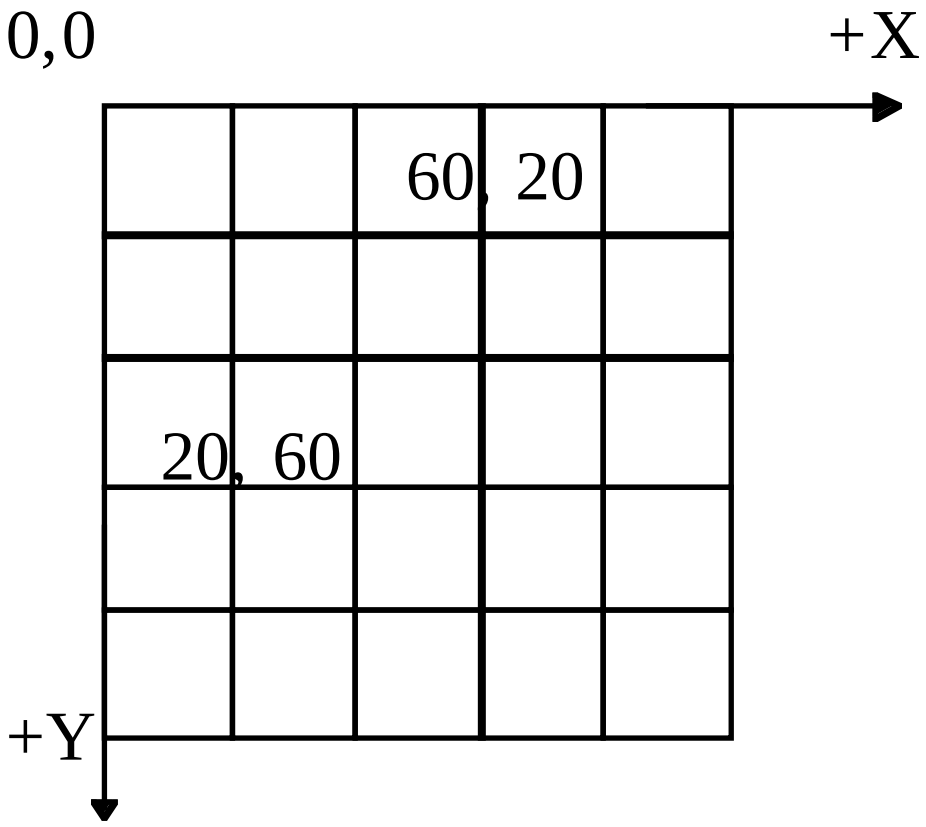
```
import java.awt.Graphics;

public class Figure extends java.applet.Applet {
   public void paint(Graphics g) {
   ...
   }
}
```

Feuille de dessin

Applet hérite de java.awt.Component un contexte graphique de type Graphics qui est la feuille de dessin.

Système de coordonnées entières



size() donne la dimension de la feuille, de type Dimension

- (0,0) le coin gauche supérieur.
- (0,Size().width) le coin droite supérieur
- (0,Size().hight) le coin gauche inférieur
- (Size().hight,Size().width) le coin droite inférieur

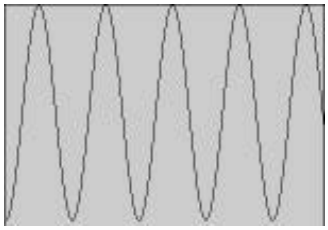
Dessiner une ligne

- Par défaut la feuille est remplie avec une couleur de fond (background)
- On dessine avec la couleur sélectionnée (par défaut le noir).
- Pour dessiner une ligne, il faut spécifier le point de départ de la ligne et son point d'arrivée.

```
public void paint(Graphics g) {
  g.drawLine(50,25,100,25);
}
```

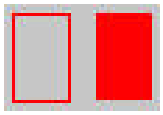


```
import java.awt.Graphics;
public class Figure extends java.applet.Applet {
  double f(double x) {
    return (Math.cos(x/10)+1) * size().height / 2;
  }
  public void paint(Graphics g) {
    for (int x = 0 ; x < size().width ; x++) {
      g.drawLine(x, (int)f(x), x + 1, (int)f(x + 1));
    }
  }
}
```



Dessiner des rectangles

```
public void paint(Graphics g) {
  g.drawRect(50,30,20,30);
  g.fillRect(80,30,20,30);
}
```

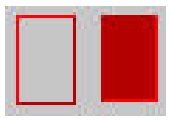


Rectangles aux coins arrondis;
il faut alors ajouter deux paramètres:
- (int) arrondiX: largeur de l'arrondi en x
- (int) arrondiY: hauteur de l'arrondi en y

```
public void paint(Graphics g) {
  g.drawRoundRect(50,70,20,30,15,15);
  g.fillRoundRect(80,70,20,30,15,20);
}
```

Rectangles en relief
Un paramètre supplémentaire booléen indique:
- false: le rectangle est en relief
- true: le rectangle est en creux

```
public void paint(Graphics g) {
  g.draw3DRect(50,110,20,30,true);
  g.fill3DRect(80,110,20,30,false);
}
```

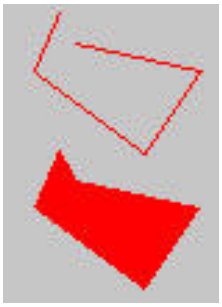


Polygones

Spécifiés par deux tableaux: coordonnées en x et coordonnées en y.
Le polygone n'est pas automatiquement fermé.
Spécifier le nombre de points à dessiner.

Dessine deux polygones.

```
public void paint(Graphics g) {
  int listeX[]={50,40,80,100,55};
  int listeY[]={150,170,200,170,160};
  int nbrXY=listeX.length;
  g.drawPolygon(listeX, listeY, nbrXY);
  int listeY2[]={200,220,250,220,210};
  g.fillPolygon(listeX, listeY2, nbrXY);
}
```



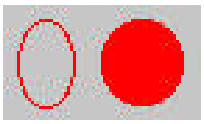
Cercles, ovales, arcs

Les formes ovales sont inscrites dans un rectangle
=> paramètres : x, y, largeur et hauteur.

Cercles => la hauteur doit égaler la largeur.

Dessine deux ovales.

```
public void paint(Graphics g) {
  g.drawOval(120,30,20,30);
  g.fillOval(150,30,30,30);
}
```



Pour dessiner des arcs
Deux paramètres:

- où doit commencer l'arc (en degré),
- sur combien de degrés on doit le dessiner.

```
public void paint(Graphics g) {
  g.drawArc(120,70,20,30,45,180);
  g.fillArc(150,70,30,30,45,-150);
}
```

Gommer, copier

Effacer une zone rectangulaire

```
g.clearRect (x, y, largeur, hauteur);
```

Copier une zone rectangulaire

nouvelle coordonnée (nx, ny)

```
g.copyArea(x, y, largeur, hauteur, nx, ny);
```

Couleurs

- classe Color pour définir et manipuler les couleurs.
- modèle de couleur sur 24 bits, 8 pour chaque couleur fondamentale (rouge, vert, bleu), => 256 x 256 x 256 = environ 16 millions de possiblités.
- le nombre de couleurs effectivement affichées dépend des possibilités du butineur et de l'écran.
- couleurs prédéfinies: white, lightGray, gray, darkGray, red, black, green, blue, yellow, magenta, cyan, pink, orange.
- définir des nouvelles couleurs et étendre la palette avec la méthode Color(R, V, B) valeurs de R, V et G comprises entre 0 et 255 (ou éventuellement entre 0.0 et 1.0), par exemple:

```
Color bleuPale= new Color(0,0,80);
```

Colorier

Méthodes héritées par Applet de la classe Component

définir la couleur de l'arrière plan
setBackground(Color)

```
setBackground(Color.white)
```

récupérer la couleur du fond.
getBackground()

imposer une couleur à tous les points déjà dessinés
setForeground(Color)

```
setForeground(Color.pink)// redessine tout
                         // en rose
```

Méthode de Graphics

redéfinir la couleur courante avec laquelle on dessine
setColor(Color)

obtenir la couleur courante
getColor()

Écrire

- La méthode drawString(String, int, int) permet de dessiner une chaîne de caractères à partir d'une coordonnée x,y.
- La couleur courante est aussi celle qui est utilisée pour écrire les caractères.

```
import java.awt.Graphics;
import java.awt.Color;

public class ApplettePolitique extends java.ap-
plet.Applet {

  public void paint(Graphics g) {
    g.setColor(Color.black);
   g.drawString("Mon premier applet politisé!", 50,
30);
    g.setColor(Color.blue);
    g.drawString("Liberté", 50, 60);
    g.setColor(Color.white);
    g.drawString("Egalité", 50, 90);
    g.setColor(Color.red);
    g.drawString("Fraternité", 50, 120);
  }
}
```

Les polices de caractères

Pour construire une police, il faut instancier un objet de la classe Font et l'initiliaser avec les paramètres suivant:

- le *nom* de la police: Helvetica, TimesRoman, Courier sont des types de caractères existant pour tous les butineurs.
- le *style* de caractères: trois constantes sont à disposition: Font.PLAIN (normal), Font.BOLD (gras), Font.ITALIC (italique). Il est possible de spécifier plusieurs styles simultanément en les additionnant.
- la *taille* des caractères: La taille est donnée en points.

Pour définir la police courante de caractères utiliées:

```
Graphics.setFont()
```

Pour interroger un objet Font: getName(), getSize(), getStyle(), isPlain(), isBold(), isItalic()

Exemple multi-polices

```
import java.awt.Graphics; import java.awt.Color;
import java.awt.Font;

public class InterPol extends java.applet.Applet {
  public void paint(Graphics g) {
    Font helvetica14Normal = new
         Font("Helvetica",Font.PLAIN,14);
    Font courier12Gras = new
         Font("Courier",Font.BOLD,12);
    Font timesRoman18Italic = new
         Font("TimesRoman",Font.ITALIC,18);
    Font timesRoman18ItalicGras = new
       Font ("TimesRoman",Font.ITALIC+Font.BOLD,18);
    g.setColor(Color.black);
    g.setFont(helvetica14Normal);
    g.drawString("Le même avec emphase!", 50, 30);
    g.setColor(Color.blue);
    g.setFont(courier12Gras);
    g.drawString("Liberté", 50, 60);
    g.setColor(Color.white);
    g.setFont(timesRoman18Italic);
    g.drawString("Egalité", 50, 90);
    g.setColor(Color.red);
    g.setFont(timesRoman18ItalicGras);
    g.drawString("Fraternité", 50, 120);
  }
}
```

Mesures des fontes

Pour effectuer du travail plus précis et plus simple sur la justification des caractères, la classe java.awt.FontMetrics donne des informations sur la taille occupée par une chaine de caractères.
Un objet de FontMetrics est créé à partir d'un objet Font p.ex.

```
FontMetrics m = getFontMetrics(g.getFont());
```

Quelques méthodes

stringWidth()	largeur d'une chaîne de caractères donnée
charWidth()	largeur d'un caractère donné
getAscent()	hauteur de la police au dessus de la ligne de base
getDescent()	hauteur de la police en dessous de la ligne de base
getLeading()	espace entre les lignes
getHeight()	hauteur total de la police

Toutes ces méthodes travaillent en points.

Exemple: centrage

```
import java.awt.*;

public class Centrage extends java.applet.Applet {
    Font helvetica18Normal = new
         Font("Helvetica",Font.PLAIN,18);
 public void writeCenter(Graphics g, String s, int
y){
    FontMetrics metriqueCourante =
         getFontMetrics(g.getFont());
    g.drawString(s,
     (size().width
         - metriqueCourante.stringWidth(s)) /2,
     y);
    }

 public void paint(Graphics g) {
    g.setColor(Color.blue);
    g.setFont(helvetica18Normal);
   writeCenter(g, "Le même avec emphase au centre!",
                30);
    writeCenter(g, "Liberté",  60);
    writeCenter(g, "Egalité",  90);
    writeCenter(g, "Fraternité",  120);
  }
}
```