Transformations Géométriques

IV) Transformations Géométriques 2D

- **Les principales transformations sont: translation, changement d'échelle, rotation, cisaillement.**

- L'utilisation du calcul matriciel permet de résoudre tous ces problèmes aisément.
- Soit (x , y) les coordonnées d'un point du plan cartésien
- (x , y) est considérée comme une coordonnée de une seule ligne sur 2 colonnes, notée **[x y]**

IV) Transformations Géométriques 2D

- On considére le produit matriciel

$$[x \quad y].\begin{bmatrix} A & B \\ C & D \end{bmatrix} = [x' \quad y']$$

- avec,

$$[Ax + Cy \quad Bx + Dy] = [x' \quad y']$$

- Donc, tout point du plan (x , y) multiplié par la matrice 2 x 2 $\begin{bmatrix} A & B \\ C & D \end{bmatrix}$ a pour transformée le point **(x’, y’)**:
  - x’ = A x + C y
  - y’ = B x + D y

IV.1) Changement d'échelle (Scale)

- Le changement d'échelle est considéré par la matrice

$$M = \begin{bmatrix} A & 0 \\ 0 & D \end{bmatrix}$$

- avec,

$$[x \quad y].\begin{bmatrix} A & 0 \\ 0 & D \end{bmatrix} = [Ax \quad Dy]$$

$$x' = A \times x$$
$$y' = D \times y$$

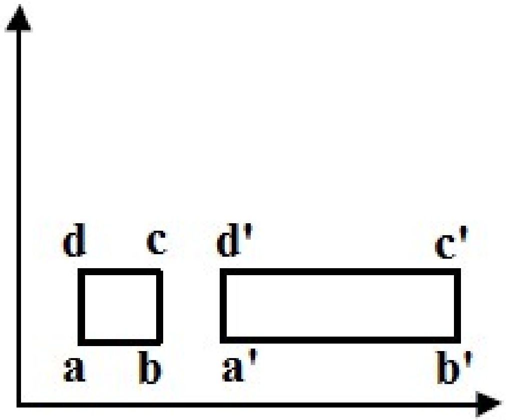
- **<u>Exemple:</u>** soit 4 points (a, b, c, d)

$$M = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$$

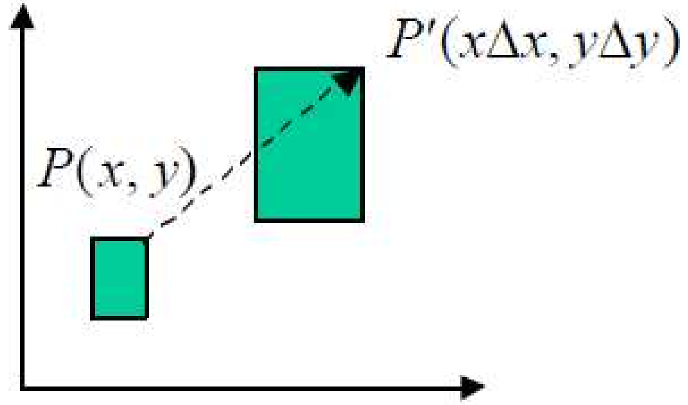
$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} * M = \begin{bmatrix} 1 & 2 \\ 2 & 2 \\ 2 & 3 \\ 1 & 3 \end{bmatrix} * \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 2 \\ 6 & 2 \\ 6 & 3 \\ 3 & 3 \end{bmatrix} = \begin{bmatrix} a' \\ b' \\ c' \\ d' \end{bmatrix}$$

IV.1) Changement d'échelle (Scale)

- Le résultat de changement d'échelle



- **En général**, la forme mise à l'échelle se trouve dans une nouvelle position:

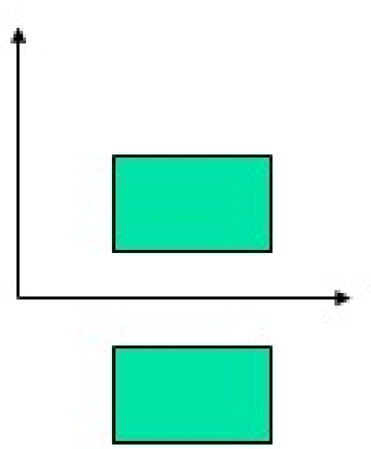


IV.2) Symétrie

- Symétrie par rapport à l'axe **(X)**

$$M = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

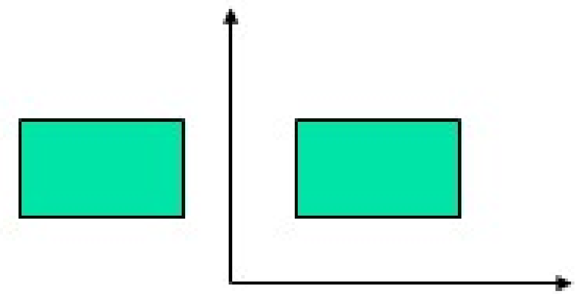
$$\begin{bmatrix} x & y \end{bmatrix} . \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} x & -y \end{bmatrix}$$



- Symétrie par rapport à l'axe **(Y)**

$$M = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

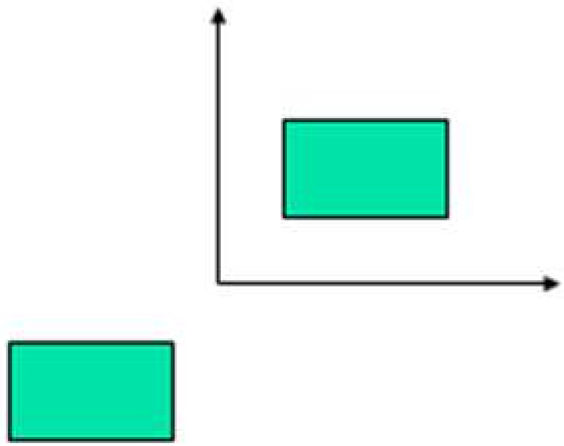
$$\begin{bmatrix} x & y \end{bmatrix} . \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -x & y \end{bmatrix}$$



IV.2) Symétrie

- Symétrie par rapport à l'origine **(O)**

$$M = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$



$$\begin{bmatrix} x & y \end{bmatrix} \cdot \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} -x & -y \end{bmatrix}$$

IV.3) Cisaillement (Shear)

- Opération qui permet de **déformer les objets**
- Soit la matrice $M = \begin{bmatrix} 1 & B \\ 0 & 1 \end{bmatrix}$
- Appliquons la matrice M au carré unitaire (a, b, c, d) pour la valeur B=2

$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 2 \\ 1 & 3 \\ 0 & 1 \end{bmatrix}$$

- On dit qu'on a effectué un cisaillement en **(y)** sur le carré unitaire.

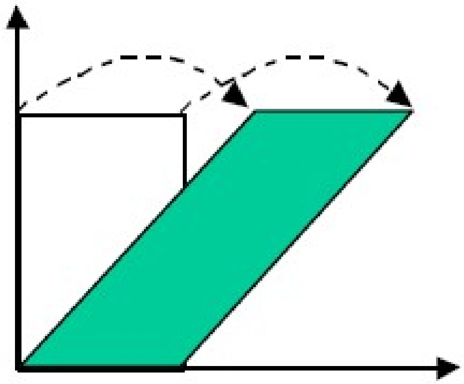
IV.3) Cisaillement (Shear)

- La matrice $M = \begin{bmatrix} 1 & 0 \\ C & 1 \end{bmatrix}$ donnera un cisaillement en **(x)**

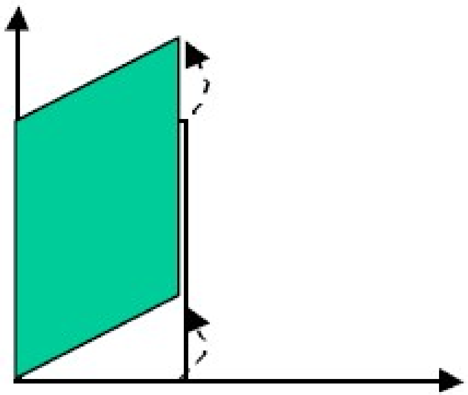
- La matrice $M = \begin{bmatrix} 1 & B \\ C & 1 \end{bmatrix}$ donnera un cisaillement dans les 2 directions x et y

IV.3) Cisaillement (Shear)

- Résultat de Cisaillement en **(x)**



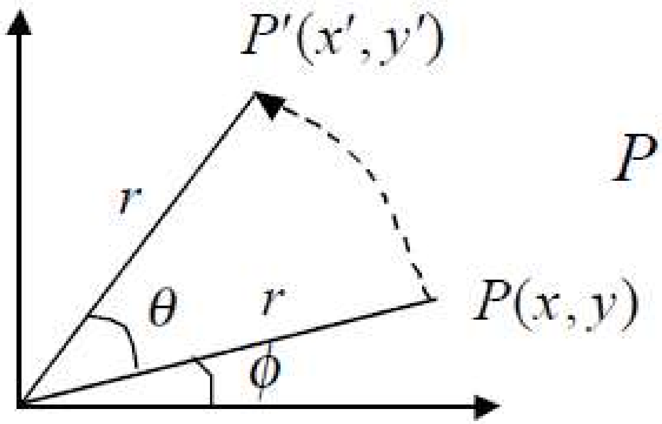
- Résultat de Cisaillement en **(y)**



IV.4) Rotation (rotate)

Rotation en 2D

$$P : \begin{cases} x = r\cos\phi \\ y = r\sin\phi \end{cases}$$



$$P' : \begin{cases} x' = r\cos(\phi+\theta) \\ \quad = \underline{r\cos\phi}\cos\theta - \underline{r\sin\phi}\sin\theta \\ \quad = x\cos\theta - y\sin\theta \\ y' = r\sin(\phi+\theta) \\ \quad = r\cos\phi\sin\theta + r\sin\phi\cos\theta \\ \quad = x\sin\theta + y\cos\theta \end{cases}$$

θ sens anti-horaire

IV.4) Rotation (rotate)

- La matrice de rotation correspondante est:

$$M_R = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

- La rotation se fait autour de l'origine avec $\mathbf{M_R}$

IV.4) Rotation (rotate)

- L'opération matricielle s'effectue par:

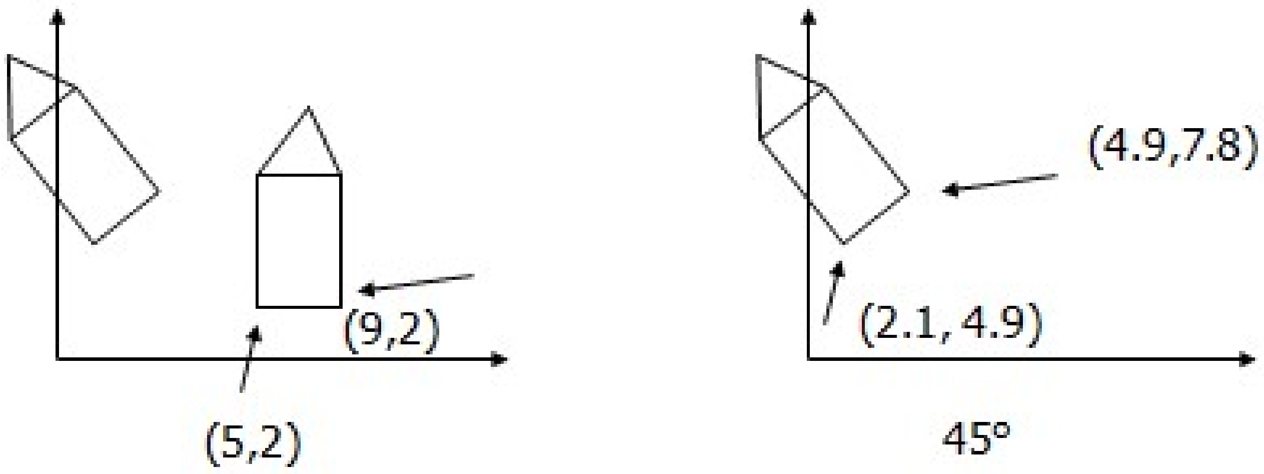
$$[x \quad y] * M_R = [x \quad y] * \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} = [x' \quad y']$$

$$[x' \, y'] = [x.\cos\theta - y.\sin\theta \quad x.\sin\theta + y.\cos\theta]$$

- **Remarque:** La rotation par rapport à un point P différent de l'origine doit subir des translations.

IV.4) Rotation (rotate)

- **Exemple: rotation par rapport à l'origine**



IV.5) Translation (translate)

- La translation permet de déplacer un point P (x , y) vers un point P' (x' , y') par des valeurs **M** et **N** telles que:

$$\begin{cases} x' = x + M \\ y' = y + N \end{cases}$$

- Il n'est pas possible d'utiliser une matrice **(2 x 2)** pour la simple opération de translation (déplacement ou mouvement d'une position initiale vers une position finale)

IV.5) Translation (translate)

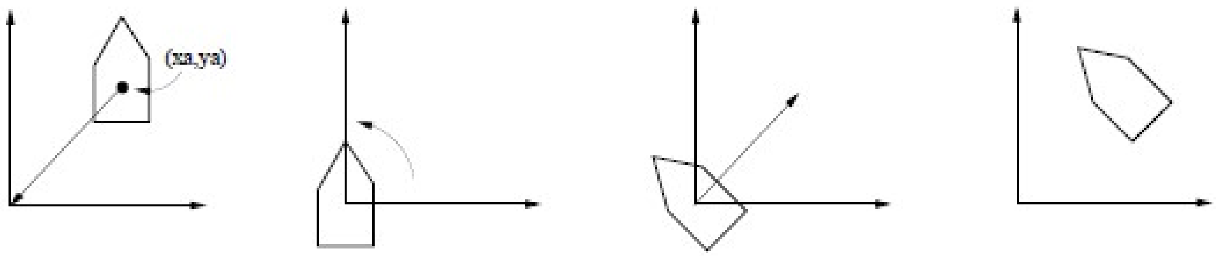
- On utilise une matrice (3 x 3) pour l'opération de translation.
- **Exemple:** translation vers un point (x'=x+M, y'=y+N):

$$\begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ M & N & 1 \end{bmatrix} = \begin{bmatrix} x+M & y+N & 1 \end{bmatrix} = \begin{bmatrix} x' & y' & 1 \end{bmatrix}$$

- La matrice (3 x 3) $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ M & N & 1 \end{bmatrix}$ est appelée matrice de coordonnées homogènes.

Compostion de transformations

- À l'aide des coordonnés homogènes, les transformations du plan se composent par simple multiplication.
- Exemple: pour la rotation autour d'un point A de coordonnées (x , y), nous appliquons 3 opérations:
  - **(A)** Translation pour ramener le point A vers l'origine
  - **(B)** Rotation d'un angle θ
  - **(C)** Translation inverse de l'origine vers le point A.



(A) **(B)** **(C)**

Composition de transformations

- **Exercice:** donner les suites de transformations nécessaires pour faire la **rotation du triangle** (a b c) d'un angle **$\theta = 30°$** autour du point P (3, 5)

- Coordonnées du triangle:
- a [7 5], b [9 5] et c [7 8].

Opérations Géométriques en Java 2D

- En Java2D on utilise la classe **Graphics2D**
- Graphics2D permet de faire les différentes transformations géométriques:

- Dans un JFrame (Panel) ou une Applet, on peut réutiliser la fonction paint:
- **public void paint (Graphics g)**
- { Graphics2D g2d = (Graphics2D)g;
- …

Opérations Géométriques en Java 2D

- <u>**Translation**</u>
- **g2d.translate (dx , dy);** ou **g2d.translate (M , N);**
- Translation ou déplacement avec une distance (dx , dy) en pixels.

$$\begin{bmatrix} 1 & 0 & 0 \\ O & 1 & 0 \\ M & N & 1 \end{bmatrix}$$

- On a: **g2d.translate (int dx , int dy);**
- Ou **g2d.translate (double dx , double dy);**

Opérations Géométriques en Java 2D

- **Changement d'Echelle**
- **g2d.scale (sx , sy);** ou **g2d.scale (A , D);**
- Changement d'échelle avec les paramètres (A , D) ou (sx , sy)
- On utilise **: g2d.scale (double sx , double sy);** ou **g2d.scale (double A , double B);**

$$M = \begin{bmatrix} A & 0 \\ 0 & D \end{bmatrix}$$

Opérations Géométriques en Java 2D

- <u>**Cisaillement**</u>
- **g2d.shear (cx , cy);** ou **g2d.shear (B , C);**
- Opération de cisaillement avec les paramètres (cx , cy) ou (B, C) en pixels.

$$M = \begin{bmatrix} 1 & B \\ C & 1 \end{bmatrix}$$

- On utilise **g2d.shear (double cx , double cy);** ou **g2d.shear (double B , double C);**

Opérations Géométriques en Java 2D

- <u>**Rotation**</u>
- **g2d.rotate (angle);** ou **g2d.rotate (θ);**
- Rotation autour de l'origine avec
  un angle **(θ)**
- On utilise **g2d.rotate (double angle);**
  ou **g2d.rotate (double θ);**

$$M_R = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

Opérations Géométriques en Java 2D

- **Rotation autour d'un point différent de l'origine**
- **g2d.rotate (angle, x, y);**
- Rotation autour du point **(x, y)** avec
  un angle **(angle)**
- On utilise **g2d.rotate (double angle, int x, int y);**

$$M_R = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

Primitives Géométriques étendues

Primitives Géométriques étendues

- **Java.awt.geom.*** contient plusieurs classes relatives aux primitives géométriques:
- comme: **Point2D, Rectangle2D, Ellipse2D, Line2D, Area** et même **Path2D** ou **GeneralPath** –ces derniers permettent de tracer une forme graphique généralisée-)
- Ex: **Rectangle2D rect = new Rectangle2D.Float(100, 120, 50, 80);**
- **Point2D pt = new Point2D.Float(150, 160);**

Primitives Géométriques étendues

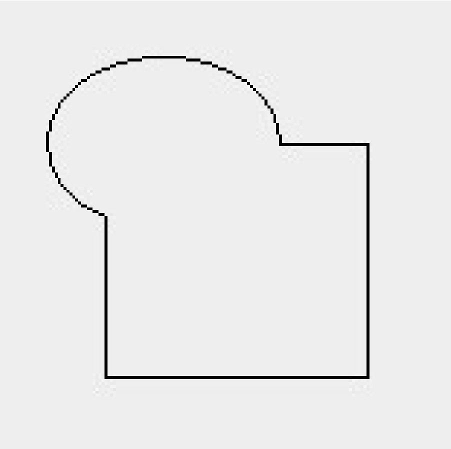
- La classe **Area** permet de combiner des formes géométriques simples pour avoir des formes complexes (**Area.add**, **intersect**, **subtract** et **exclusiveOr**).
- Exemple: le code ci-après permet de combiner une ellipse avec un rectangle pour obtenir une forme plus complexe:

Primitives Géométriques étendues

- *Graphics2D g2 = (Graphics2D) g;*
- ***Ellipse2D** oval = **new** Ellipse2D.Float(100, 200, 80,60);*
- ***Rectangle2D** rect = **new** Rectangle2D.Float(120, 230, 90, 80);*
- ***Area** forme = **new** Area(oval);*
- ***forme.add(new Area(rect));*** *// alternatives*
- ***forme.intersect(new Area(rect));*** *// à choisir*
- *g2.**draw(forme)**; **ou** g2.**fill(forme)**;:*

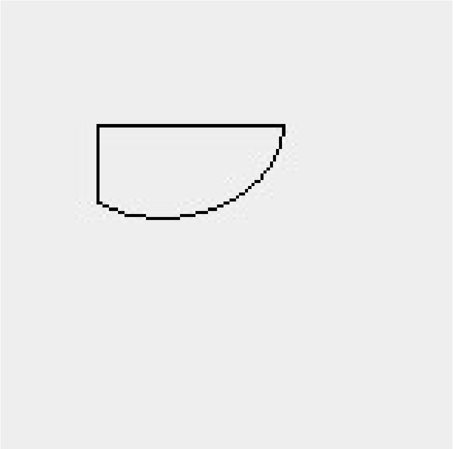
Primitives Géométriques étendues

- *Graphics2D g2 = (Graphics2D) g;*
- ***Ellipse2D** oval = **new** Ellipse2D.Float(100, 200, 80,60);*
- ***Rectangle2D** rect = **new** Rectangle2D.Float(120, 230, 90, 80);*
- ***Area** forme = **new** Area(oval);*
- ***forme.add(new Area(rect));***
- *g2.**draw(forme)**; **ou g2.fill(forme);**:*



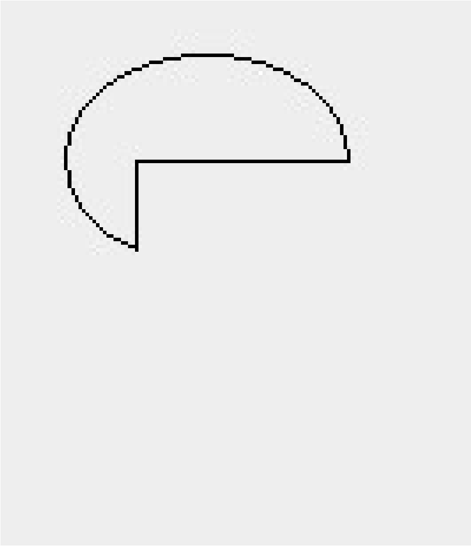
Primitives Géométriques étendues

- *Graphics2D g2 = (Graphics2D) g;*
- ***Ellipse2D** oval = **new** Ellipse2D.Float(100, 200, 80,60);*
- ***Rectangle2D** rect = **new** Rectangle2D.Float(120, 230, 90, 80);*
- ***Area** forme = **new** Area(oval);*
- ***forme.intersect(new Area(rect));***
- *g2.**draw(forme)**; **ou g2.fill(forme)**;:*



Primitives Géométriques étendues

- *Graphics2D g2 = (Graphics2D) g;*
- ***Ellipse2D** oval = **new** Ellipse2D.Float(100, 200, 80,60);*
- ***Rectangle2D** rect = **new** Rectangle2D.Float(120, 230, 90, 80);*
- ***Area** forme = **new** Area(oval);*
- ***forme.subtract(new Area(rect));***
- *g2.**draw(forme)**; **ou g2.fill(forme)**;:*



Classe AffineTransform

- Les transformations géométriques (**Graphics2D.scale, translate, rotate ou Graphics2D.shear (cx, cy)**) peuvent aussi être réalisées avec la classe **AffineTransform**
- On peut faire une série d'opérations géométriques avec la même classe **AffineTransform**
- Le code suivant qui combine une ellipse avec un rectangle introduit la classe **AffineTransform** pour réaliser une transformation géométrique

AffineTransform (exemple de code)

- Graphics2D g2 = (Graphics2D) g;
- Ellipse2D oval = new Ellipse2D.Float(100, 200,80, 60);
- Rectangle2D rect = new Rectangle2D.Float(120, 230, 90, 80);
- **Area forme = new Area(oval);**
- forme.add(new Area(rect)); // ou forme.intersect(new Area(rect));
- g2.draw(forme);

AffineTransform (exemple de code)

- **AffineTransform at = new AffineTransform();**
- **at.translate**(100, 100);// Translation de (x=100, y=100)
- **at.rotate (0.5)**; // Rotation d’un angle de 0,5 radians
- **forme.transform(at);** //appliquer la translation suivie de la rotation
- **g2.draw(forme);** // Dessiner la forme, ou utiliser g2.fill(forme)

AffineTransform (exemple de code)

- **AffineTransform at2 = new AffineTransform();**
- **at2.translate**(-10, -50);// Translation de (x=-10, y=-50)
- **at2.shear (1, 0)**; // cisaillement de 1 en x
- **at2.scale (2, 1.5)**; // zoom (changem. d'échelled [2 en x, 1.5 en y])
- **forme.transform(at2);** //applique la translation suivie d'un cisaillement suivie d'un changement d'échelle
- **g2.fill(forme);** // Dessiner la forme remplie