Dibujando en un JPanel

Definir una subclase de JPanel y redefinir el método public void paintComponent(Graphics g) . Este método llamará inicialmente a super.paintComponent() y luego incluirá el código necesario para efectuar el dibujo.

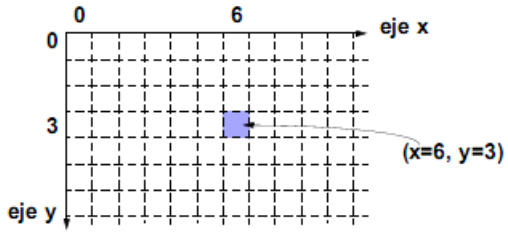
El código que incluyamos en este método se ejecutará cada vez que el panel necesita ser dibujado en la pantalla. El sistema invoca automáticamente a este método cada vez que sea necesario.

```
public PanelDibujo extends JPanel
{
        ..............
        public void paintComponent(Graphics g)
        {
                super.paintComponent(g);
                . . . // instrucciones para efectuar el dibujo
        }

 ..............

}
```

super.paintComponent(g); // llamada a paintComponent() de JPanel para limpiar el panel antes de dibujar

> paintComponent(g) **nunca debe invocarse directamente**, hay que hacerlo a través de repaint() (public void repaint()). Si el panel necesita volver a ser dibujado porque ha habido cambios se lo comunicaremos al sistema llamando a repaint();

Sistema de coordenadas:



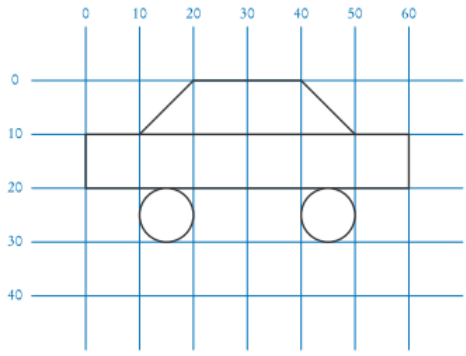


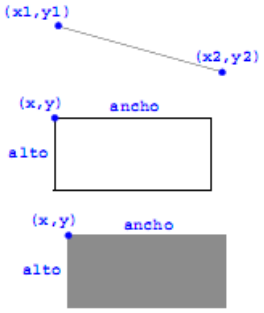
Figure 7 Using Graph Paper to Find Shape Coordinates

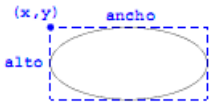
El objeto *g* de tipo Graphics (**java.awt.Graphics**) representa el contexto gráfico necesario para poder efectuar cualquier dibujo. La clase Graphics proporciona métodos para dibujar figuras, texto, imágenes.

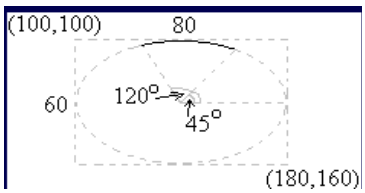
Clase java.awt.Graphics

Métodos	Descripción
void drawString(String str, int x, int y)	Dibuja una cadena de texto en la posición *(x,y)*
void drawLine(int x1, int y1, int x2, int y2)	Dibuja una línea con origen en *(x1,y1)* y final en *(x2,y2)*
void drawRect(int x, int y , int alto, int ancho)	Dibujo de un rectángulo partir de *(x,y)* con el alto y ancho especificado

void fillRect(int x, int y , int alto, int ancho)	Rectángulo relleno
void drawOval(int x, int y , int ancho, int alto)	Dibujo de un óvalo
void drawArc(int x, int y , int ancho, int alto, int anguloInicio, int anguloFin))	Dibujo de un arco
void setColor(Color c)	Selección de un color. A partir de ese momento lo que se dibuje con *g* será de ese color
Color getColor())	Obtener el color







g.drawArc(100, 100, 80, 60,45,120);

```
g.setColor( Color.red);
g.drawLine( x1, y1, x2, y2);

g.setColor( Color.green);
g.fillRect( x, y, ancho, alto);

g.setColor(new Color(122, 223, 167));
g.drawString( x, y, "mensaje");
```

Clase java.awt.Color

Table 1 Predefined Colors

Color	RGB Value
Color.BLACK	0, 0, 0
Color.BLUE	0, 0, 255
Color.CYAN	0, 255, 255
Color.GRAY	128, 128, 128
Color.DARKGRAY	64, 64, 64
Color.LIGHTGRAY	192, 192, 192
Color.GREEN	0, 255, 0
Color.MAGENTA	255, 0, 255
Color.ORANGE	255, 200, 0
Color.PINK	255, 175, 175
Color.RED	255, 0, 0
Color.WHITE	255, 255, 255
Color.YELLOW	255, 255, 0

Color magenta = new Color(255, 0, 255);

Clase java.awt.Font

```
Font font = new Font("Georgia", Font.BOLD, 26);
g.setFont(font);
```

JComponent (Métodos comunes a todos los componentes incluido JPanel)

Método o constructor	Descripción
void setBackground(Color c)	Establece el color de fondo del componente
void setForeground(Color c)	Color de la superficie
void setFont(Font fuente)	Establece tipo de fuente
void setEnabled(boolean)	Habilita / deshabilita el componente
void setSize(Dimension d)	Establece el tamaño del componente
vois setPreferredSize(Dimension d)	Establece el tamaño preferido del componente
void setVisible(boolean)	Hacer o no visible el componente
void addXXXListener(XXXListener)	Añade un oyente de un tipo específico

Eventos de ratón (mouse events)

```
public interface MouseListener
{
        void mousePressed(MouseEvent event);
          // llamado cuando un botón del ratón se ha pulsado sobre un componente
        void mouseReleased(MouseEvent event);
          // llamado cuando un botón del ratón ha sido liberado sobre un componente
        void mouseClicked(MouseEvent event);
              // llamado cuando  el botón del ratón se presiona y se libera sin movimiento del ratón entre ambas
             //acciones
        void mouseEntered(MouseEvent event);
          // llamado cuando el ratón entra en un componente
        void mouseExited(MouseEvent event);
          // llamado cuando el ratón sale de un componente
}
```

Cada uno de estos métodos recibe un parámetro *event* de tipo `MouseEvent` que guarda información acerca del evento ocurrido.

La clase MouseEvent

```
public int getX()
public int getY()
```

public Point getPoint() - devuelve la posición *x,y* relativa al componente que generó el evento

public int getClickCount() - devuelve el nº de *clicks* de ratón asociados al evento

Eventos de ratón y Clases adaptadoras

```
public class OyenteRaton implements MouseListener
{
        public void mouseClicked(MouseEvent event)
        {
            // código
        }
        // cuatro métodos que no hacen nada
        public void mouseEntered(MouseEvent event) { }
        public void mouseExited(MouseEvent event) { }
        public void mousePressed(MouseEvent event) { }
        public void mouseReleased(MouseEvent event) { }
}
```

Mejor usar clases adaptadoras.

```
public class OyenteRaton extends MouseAdapter
{
        public void mouseClicked(MouseEvent event)
        {
            // código
        }
}
```

Eventos de teclado y clases adaptadoras. La clase KeyEvent.

```
public interface KeyListener
{
        public void keyPressed(KeyEvent e);  //  llamado al pulsar  una tecla
        public void keyReleased(KeyEvent e);  // llamado al liberar una tecla
        public void keyTyped( KeyEvent e);
          // llamado al teclear un carácter  o  una combinación de teclas que producen un carácter

}
```

KeyEvent.VK_SHIFT	Shift
KeyEvent.VK_LEFT	cursor izquierda
KeyEvent.VK_RIGHT	cursor derecha
KeyEvent.VK_UP	cursor arriba
KeyEvent.VK_DOWN	cursor abajo
KeyEvent.VK_ENTER	enter
KeyEvent.VK_A, KeyEvent.VK_B, ...	carácter 'A' 'B' .....

Cada tecla en el teclado tiene un código numérico asociado.
El objeto *e* de tipo KeyEvent incluye información sobre el evento producido y , por tanto, sobre la tecla pulsada.
El código asociado a la tecla pulsada se obtiene a través del método e.getKeyCode().

Java proporciona constantes estáticas asociadas a cada tecla definidas en la clase KeyEvent. Todas las constantes empiezan por VK (*Virtual KeyBoard)*.

```
public class OyenteTeclado  implements KeyListener
{
        public void keyPressed(KeyEvent e)
        {
            // código  para procesar la tecla
             if (e.getKeyCode() == KeyEvent.VK_UP)
             {
                    ……..
             }
             else if (e.getKeyCode() == KeyEvent.VK_DOWN)
             {
                    ……..

             }
             else if (e.getKeyCode() == KeyEvent.VK_LEFT)
             {
                    ……..

             }
        }
         // dos métodos que no hacen nada
       public void keyReleased(KeyEvent e);
       public void keyTyped( KeyEvent e);
}
```

También se puede hacer de esta forma,

```
public class OyenteTeclado  implements KeyListener
{
        public void keyPressed(KeyEvent e)
        {
            // código  para procesar la tecla
             String key = KeyStroke.getKeyStrokeForEvent(event).toString();
             key = key.replace("pressed ", "");
             if (key.equals("RIGHT"))
             {
                    ……..
             }
              else if (key.equals("LEFT"))
             {
                    ……..
             }
             else if (key.equals("DOWN"))
             {
                    ……..
             }

        }
         // dos métodos que no hacen nada
       public void keyReleased(KeyEvent e);
       public void keyTyped( KeyEvent e);
}
```

Convierte el evento e en una descripción textual de la tecla pulsada, por ej, “pressed LEFT”. Después se elimina de la cadena “pressed” y lo que queda es un string como “LEFT”, “UP”, “A”, ...

Mejor usar clases adaptadoras.

```
public class OyenteTeclado  extends KeyAdapter
{
		public void keyPressed(KeyEvent e)
		{
			// código
		}

}
```