HSA2 Graphics Console

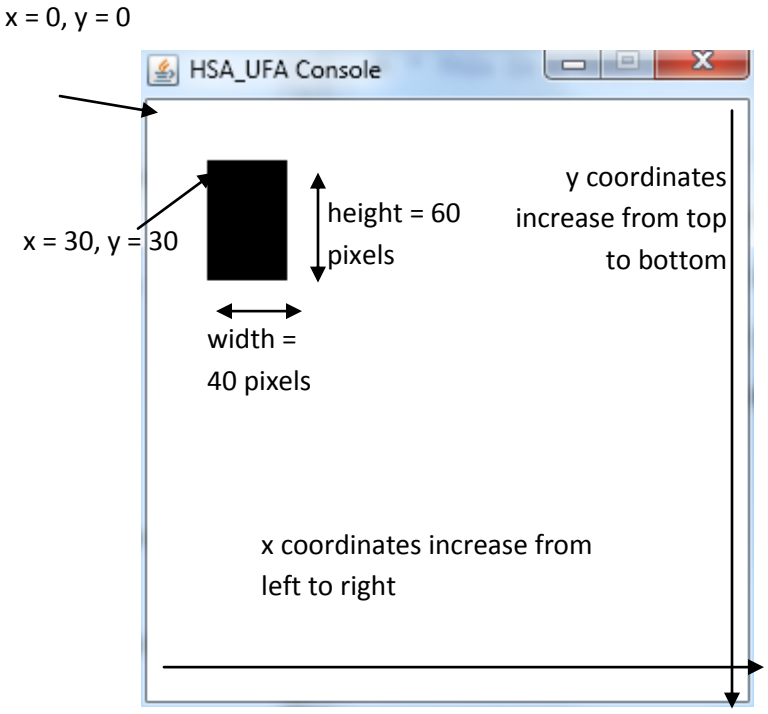
Introduction

You already know that the Console window can output text, laid out in rows and columns of **characters** (any letter, number, punctuation mark, space, etc. is a character).

But the Console can also output graphics, and for that you have to think of it as laid out in rows and columns of **pixels** (a pixel is the smallest dot of color that can appear on a computer screen).

Graphics Output

For graphics, the console is divided into **x and y coordinates**. Each x and y coordinate identifies a single pixel. The top left corner is x=0 and y=0. Whenever you draw a shape, you usually have to specify an x and y starting position, and at least a width and height for the shape, in pixels.



Basic Graphics Methods

1. Colors

setColor (Color)

Sets the drawing color for all text and graphics output methods.

setBackgroundColor (*Color*)

Sets the background color for all text and graphics output methods.

2. Other useful methods

setAntiAlias(true or **false)**

Turns AntiAliasing on or off. This makes all lines smoother when drawing and using drawstring()

setStroke(size)

Enter an integer to set the size of the drawing stroke. Does not affect text output nor drawstring().

setResizable(true or **false)**

This enables or disables resizing the graphics console by dragging the mouse. (Not really a graphics method)

showDialog(message, title)

Creates a popup message box (dialog) with the message and title specified. The program stops until OK is pressed.

showInputDialog(message, title)

This makes a dialog box that allows a line of text to be typed in. The dialog box has the message and title specified. The program stops until OK is pressed. The following code shows how to handle “cancel” and empty strings:

```
String name = c.showInputDialog("What is your name?", "");

if(name == null){                    //handle CANCEL option
    System.out.println("Cancel pressed");
    System.exit(0); //or do something else
}
if (name.equals("")) name = "No Name";      //handle OK option with no text
```

3. Drawing methods

drawLine (x1, y1, x2, y2)

Draws a line from (*x1*, *y1*) to (*x2*, *y2*).

drawRect (x, y, width, height)

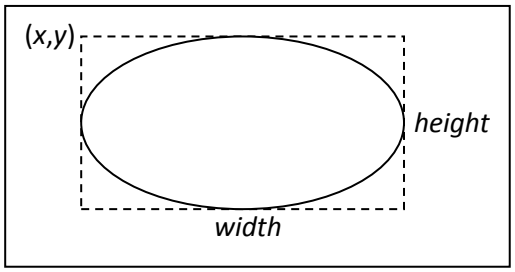
Draws a rectangle with upper-left corner at (*x*, *y*) with width of *width* and height of *height*.

fillRect (x, y, width, height)

Draws a filled rectangle with upper-left corner at (*x*, *y*) with width of *width* and height of *height*.

drawOval (x, y, width, height)

Draws an oval. The oval is inscribed in the rectangle defined by the upper-left corner (*x*, *y*) with width of *width* and height of *height*.



fillOval (x, y, width, height)

Draws a filled oval. The oval is inscribed in the rectangle defined by the upper-left corner (*x*, *y*) with width of *width* and height of *height*.

drawPolygon(int[] x, int[] y, int n)
fillPolygon(int[] x, int[] y, int n)

Draws (or fills) a polygon on the drawing area. The polygon is created on the fly by a list of x,y values for the vertices and 'n' stating how many vertices there are.

drawPolygon(Polygon p)
fillPolygon(Polygon p)

Draws (or fills) a polygon on the drawing area. The polygon p is created ahead of time. The following code shows how to draw a triangle:

```
Polygon p = new Polygon();
p.addPoint(50, 100);
p.addPoint(150, 310);
p.addPoint(350, 70);
c.drawPolygon(p);
```

★The following (maple leaf and star) are NOT standard Java Swing commands:

drawMapleLeaf (x, y, width, height)

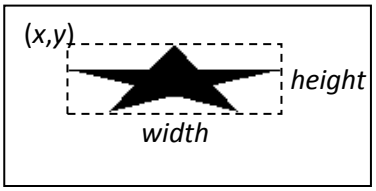
Draws a maple leaf. The maple leaf is inscribed in the rectangle defined by the upper-left corner (*x*, *y*) with width of *width* and height of *height*.

fillMapleLeaf (x, y, width, height)

Draws a filled maple leaf. The maple leaf is inscribed in the rectangle defined by the upper-left corner (*x*, *y*) with width of *width* and height of *height*.

drawStar (x, y, width, height)

Draws a star inscribed in the rectangle defined by the upper-left corner (*x*, *y*) with width of *width* and height of *height*.



fillStar (x, y, width, height)

Draws a filled star inscribed in the rectangle defined by the upper-left corner (*x*, *y*) with width of *width* and height of *height*.

Better Text Output

drawString (str, x, y)

Draws the string *str* at the starting point (*x*, *y*). The *y* coordinate is the bottom of the text.

setFont (Font)

Sets the font for the drawString method, but does not affect print or println. See the font handout.

Advanced Graphics Methods

drawRoundRect (x, y, width, height, arcWidth, arcHeight)

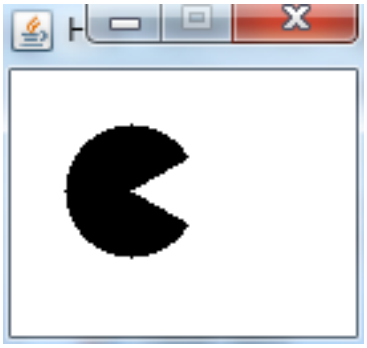
Draws a rectangle with rounded corners with upper-left corner at (*x*, *y*) with width of *width* and height of *height*. *arcWidth* and *arcHeight* are the width and height of the ellipse used to draw the rounded corners.

fillRoundRect (x, y, width, height, arcWidth, arcHeight)

Draws a filled rectangle with rounded corners with upper-left corner at (*x*, *y*) with width of *width* and height of *height*. *arcWidth* and *arcHeight* are the width and height of the ellipse used to draw the rounded corners.

drawArc (x, y, width, height, startAngle, arcAngle)

Draws an arc. The arc is inscribed in the rectangle defined by the upper-left corner (*x*, *y*) with width of *width* and height of *height*. It starts at *startAngle* degrees and goes counterclockwise for *arcAngle* degrees.



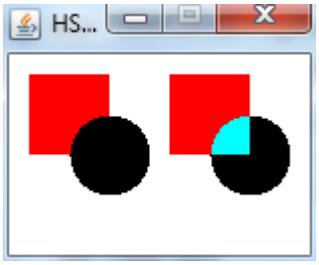
```
c.fillArc (20, 20, 50, 50,
```

fillArc (x, y, width, height, startAngle, arcAngle)

Draws a filled arc. The arc is inscribed in the rectangle defined by the upper-left corner (*x*, *y*) with width of *width* and height of *height*. It starts at *startAngle* degrees and goes counterclockwise for *arcAngle* degrees.

setPaintMode ()

All drawing will now cover or overlay whatever is underneath it (ie. opposite to XORmode) .



setXORMode (backgroundColor)

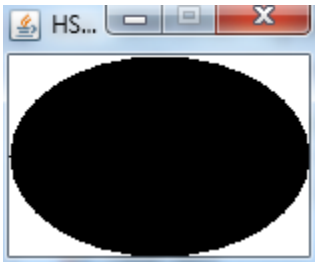
Any time you draw over something, it will show through. (Technically, the new shape is XOR'd with the background.) You must specify the background color to stop it from showing through when you draw.

getDrawHeight ()

Returns the height of the drawing area (graphics console) in pixels

getDrawWidth ()

Returns the width of the drawing area in pixels.
For example, you could use *getWidth()* in place of the width number in a call to *drawOval()*, and the oval will be as wide as the screen.




```
c.fillOval(0, 0, c.getDrawWidth(),
c.getDrawHeight());
```