HSA2 Graphics and Drawing Commands

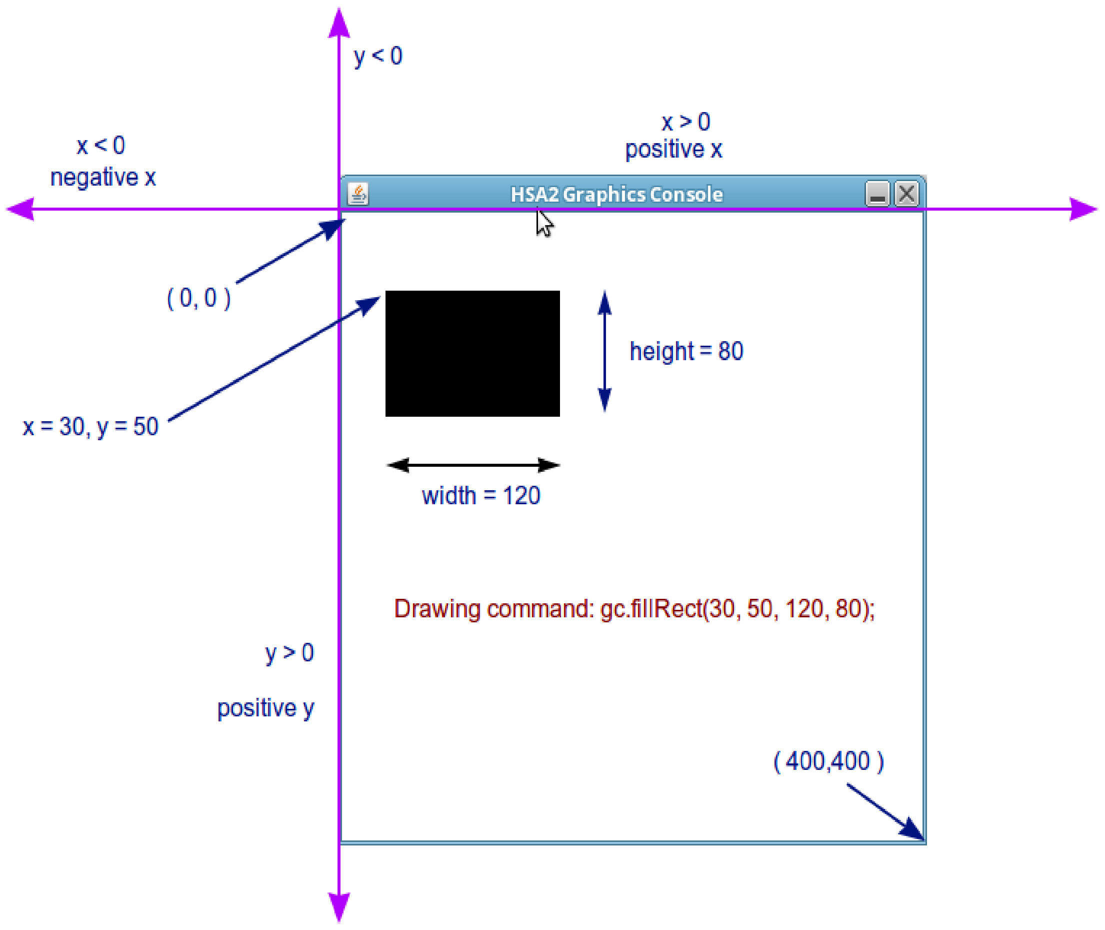
1. Overview of computer graphics
2. Common drawing commands (you should know all of these)
3. Other useful graphics commands (again, you'll use most of these)
4. Less common drawing commands (can be omitted)
5. Extra HSA2-only drawing commands (can be omitted).

► 1. Computer Graphics

In all computer graphics, the screen has **x and y coordinates**. Each x and y coordinate identifies a single pixel.
Note that the top left corner is x=0 and y=0. The x-axis is normal as in math class,
but the y-axis starts at the top of the screen and increases as it goes down. Positive y is downwards.

Here is an example showing the x,y axes and the results of the commands:

```
GraphicsConsole gc = new GraphicsConsole( 400, 400);
gc.fillRect(30, 50, 120, 80);
```



Most shapes require that you specify an x and y starting position (always the top-left corner),
and a width and height for the shape, in pixels.

► 2. Common Drawing Commands

★All of these use the current foreground colour, the current font, and the current line size.

● Draw a line: **gc.drawLine** (x1, y1, x2, y2) ;

Draws a line between the two points (*x1*, *y1*) to (*x2*, *y2*).
This is the one command that doesn't use width and height.

● Draw a rectangle or square: **gc.drawRect** (x, y, width, height) ;

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

● Fill in a rectangle **gc.fillRect** (x, y, width, height) ;

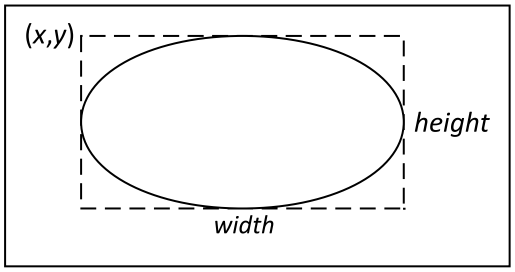
Draws a filled rectangle with upper-left corner at (*x*, *y*) with width of *width* and height of *height*.

★ TIP: To outline a rectangle, do a fillRect followed by drawRect (using different colours)

```
gc.setColor(Color.GREEN);
gc.fillRect(50,50,100,100);
gc.setColor(Color.YELLOW);
gc.drawRect(50,50,100,100);      //we now have a green square with a yellow outline.
```

● Draw an oval or circle: **gc.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*.



● Fill in an oval or circle **gc.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*.

● Write text on the screen **gc.drawString** (str, x, y);

Draws the string *str* at the starting point (*x*, *y*) using the current font and foreground colour.
The *y* coordinate is the <u>bottom</u> of the text (unlike all other shape drawing).

```
Example :   gc.drawString( "Lives = " + player.lives , 50,70);
```

● Draw or fill a polygon (first method – *I prefer this one*)

gc.drawPolygon(Polygon p); **gc.fillPolygon(Polygon p);**

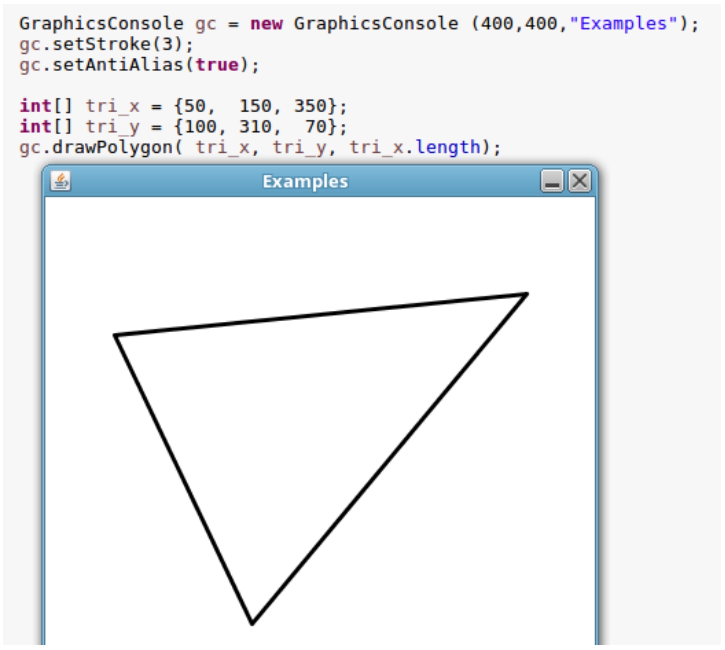
Draws (or fills) a polygon on the drawing area. The polygon p is created ahead of time. The following code shows how to draw a triangle:

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

● Draw or fill a polygon (second method)

gc.drawPolygon(int[] x, int[] y, int n); **gc.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.



```
GraphicsConsole gc = new GraphicsConsole (400,400,"Examples");
gc.setStroke(3);
gc.setAntiAlias(true);

int[] tri_x = {50,  150, 350};
int[] tri_y = {100, 310,  70};
gc.drawPolygon( tri_x, tri_y, tri_x.length);
```

► 3. Other useful graphics methods

There is a separate document explaining how to change colours and fonts.

● Change line thickness: **gc.setStroke**(size);

Enter an integer to set the size of the drawing stroke.
Does not affect text output nor drawstring().
The default value is 1.

● Turns AntiAliasing on or off: **gc.setAntiAlias(true** or **false);**

This makes all diagonal and curved lines smoother when drawing graphics.
It also smooths text when when using drawstring().
You should probably always use this unless all your lines are vertical and horizontal.
The default value is false.

● Allow window size to be changed **gc.setResizable(true** or **false);**

This enables or disables resizing the graphics console by dragging the mouse.
If you do this, you'll have to recalculate where you're drawing things based on the window size.
The default value is false.

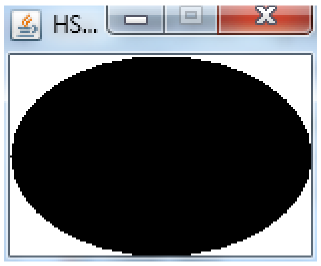
● Centre window on screen **gc.setLocationRelativeTo(null);**

● **gc.getDrawWidth ();** and **gc.getDrawHeight** ();

This returns the width and the height of the drawing area (graphics console) 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:

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



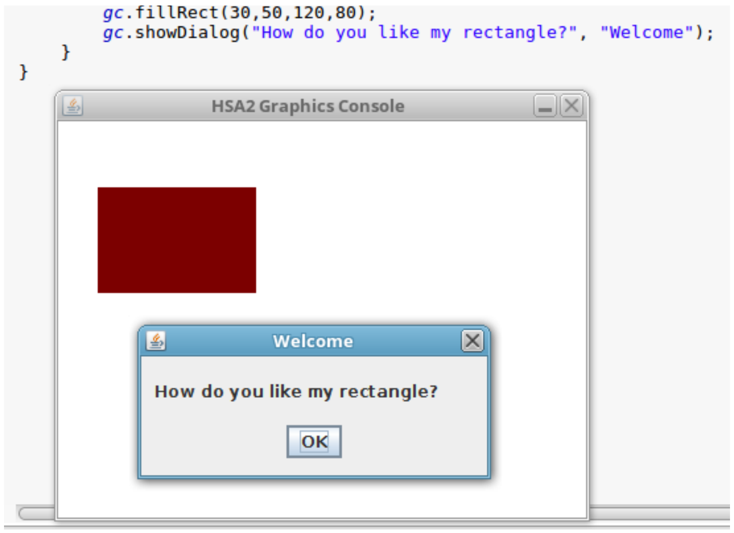
These methods <u>are only useful</u> if you enable windows to be resized.
Otherwise you already know the window width and height and there's no need to ask what they are.
We typically set global constants for these values:

```
static final int WINW = 1024;
static final int WINH = 900;
GraphicsConsole gc =  new GraphicsConsole(WINW, WINH);
```

● Popup message box **gc.showDialog(String message, String title);**

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

★ In graphics programs you should be using this or gc.drawString to print messages to the user, not System.out.println().



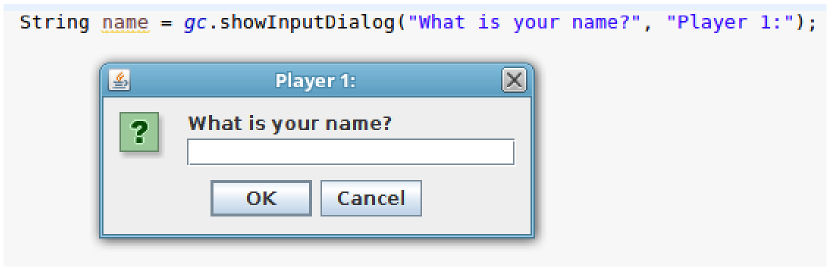
```
        gc.fillRect(30,50,120,80);
        gc.showDialog("How do you like my rectangle?", "Welcome");
    }
}
```

Example : `gc.showDialog( "Thank you for playing" , "Game Over");`

● Popup message box getting user input **gc.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 = gc.showInputDialog("What is your name?", "Player 1:");
if(name == null){              //handle CANCEL option
    System.out.println("Cancel pressed");
    System.exit(0);  //or do something else
}
//handle OK option with no text
if (name.equals("")) name = "No Name";
```



```
String name = gc.showInputDialog("What is your name?", "Player 1:");
```

If you ask for a number (e.g. how many players there are), the answer is still stored as a string and you’ll have to convert it to a number before using it.

★ In graphics programs you should be using this to get input from the user, not System.out.println() and Scanner which are console based.

► 4. Advanced Graphics Methods (infrequently used)

★ drawArc() and fillArc() are the most useful ones here.

● **gc.drawRoundRect** (x, y, width, height, arcWidth, arcHeight);

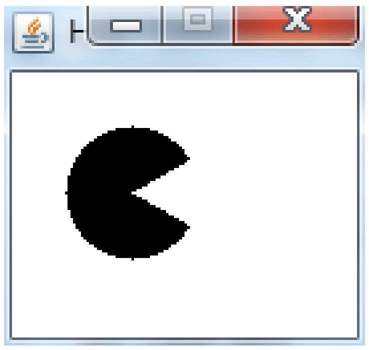
Draws a rectangle with rounded corners with upper-left corner at (*x*, *y*) with width of *width* and height of *height*. *arcWidth* and *arcHeight* are the width and height of the ellipse used to draw the rounded corners.

● **gc.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.

● **gc.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.



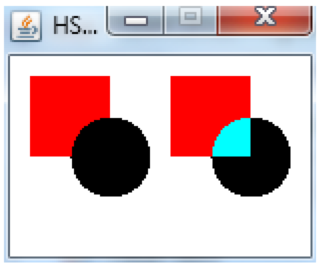
```
gc.fillArc (20, 20, 50, 50,

30, 300);
```

● **gc.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.

● **gc.setPaintMode** ();

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



● **gc.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.

► 5. Extra HSA-only drawing commands

The following (maple leaf and star) are NOT standard Java Swing commands.
They will ONLY work if hsa2 is imported. There's probably no point using them.
They are only kept for backwards compatibility

- **gc.drawMapleLeaf** (x, y, width, height);

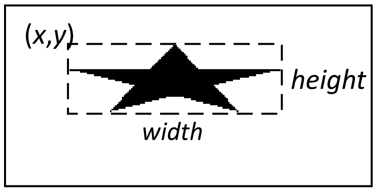
  Draws a maple leaf. The maple leaf is inscribed in the rectangle defined by the upper-left corner (*x*, *y*) with width of *width* and height of *height*.

- **gc.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*.

- **gc.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*.



- **gc.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*.