Colours in Java

Colours are described by Color objects, that is, objects created by the Color class.

►You must **import java.awt.Color;** in order to use Color in Java.◄

There are about 14 built-in colours that you can use right away. They are
Color.BLACK, Color.BLUE, Color.CYAN, Color.DARK_GRAY, Color.GRAY, Color.GREEN, Color.LIGHT_GRAY, Color.MAGENTA, Color.ORANGE, Color.PINK, Color.RED, Color.WHITE, AND Color.YELLOW.

Strangely enough, lowercase versions of these colours exist too. Color.red is the same as Color.RED, but this doesn't make sense since they are all constants and cannot be changed. The convention in Java is that constants be ALLCAPS.

When you change the colour, you do not change the colour of something already on the screen. Instead it works as if you are picking up a new coloured pen. Everything you draw from now on will be in the new colour. So what you do is, (i) set the colour, then (ii) draw something.

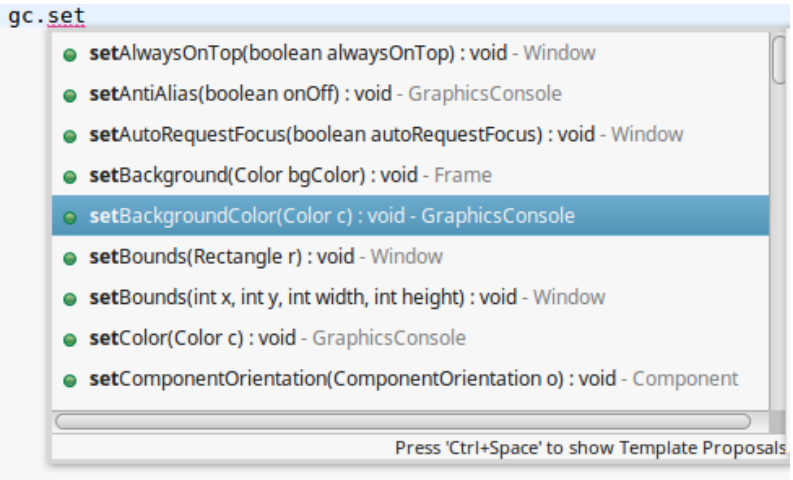
► Color Related Commands

● Set the foreground colour : **gc.setColor(Color.RED);**

● Set the background colour : **gc.setBackgroundColor(Color.BLUE);**
gc.clear();

★ *You have to do* **gc.clear();** *after you set the background colour. This clears the screen, and then fills it with the current background colour so that you can draw on it.*

★ There are two background colour commands (as you can see in the image below).
setBackground() is part of the Frame class. **Do not use it.**
It won't work (unless you're using Swing and Jpanels). Use **setBackgroundColor()** instead.



★ When you use commands (methods) following **gc.** , always pick methods that show that they are part of the GraphicsConsole class. Most other methods will not work properly with HSA2.

Creating your own colours

Colours are created by specifying 3 numbers as parameters.

For example *Color bgColour = new Color(55,66,77);* *//this is now a variable called bgColour that is a Color.*

Once you've made a colour, you can use it anywhere you want in your program (assuming that it's a global variable) where a colour object can be used.

```
final static Color bgColour = new Color(55,66,77);

gc.setBackgroundColor(bgColour);
```

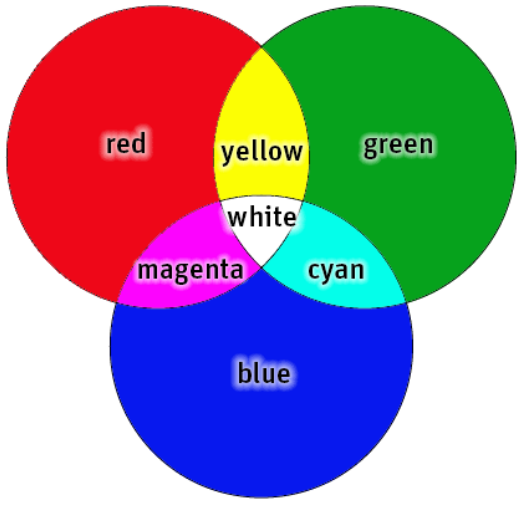
The three integers specify how much of the colors **red**, **green** and **blue** to mix.
➔ 0 is the minimum and 255 is the maximum amount. ➔
Anything else will cause an error.
new Color(0, 0, 0) is black,
new Color(255, 255, 255) is white,
new Color(255, 0, 0) is bright red *(100% red, zero green, zero blue)*
new Color(100, 0, 0) is dull red, and so on.

★ You can use a colour without having to assign it to a variable, like this:

```
gc.setColor( new Color( 100, 255, 100) );
```

Colour mixing follows the rules for lights (TV, computer monitors, stage lighting).
Thus, the three primary colours are **red**, **green** and **blue**.

R + B = magenta
B + G = cyan
R + G = yellow
R + G + B = white

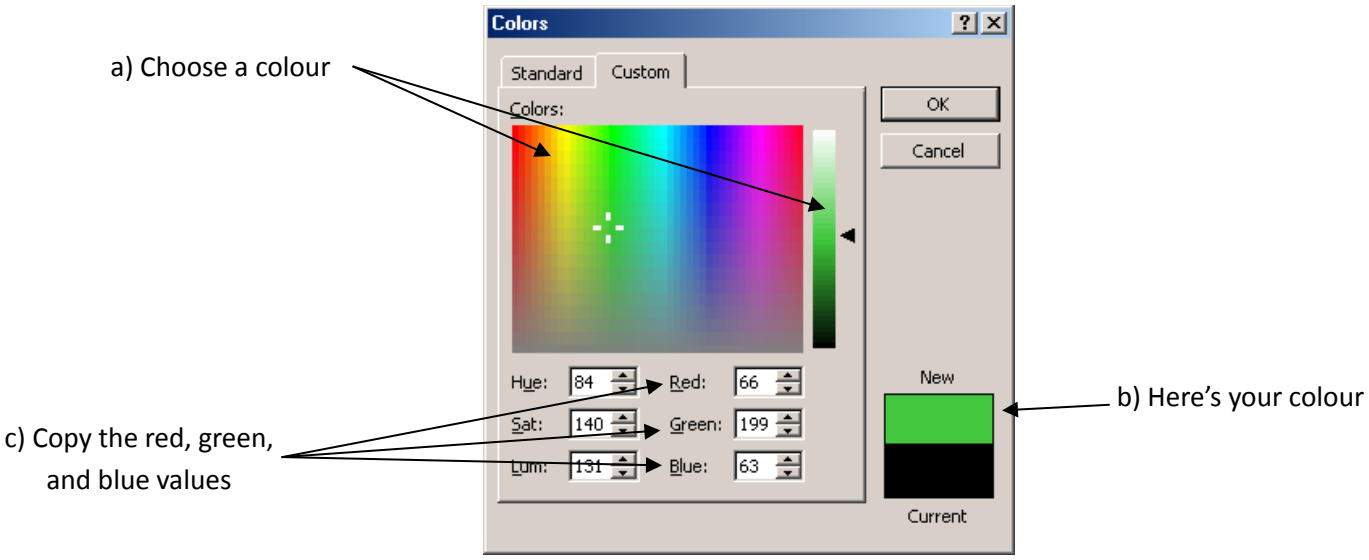


Combining Red, Green and Blue

How do you find out the red/green/blue values for a good colour?
How do you create your own colours?

1. There are colour charts online. Note: often the colour values are given in hexadecimal (00-FF) instead of decimal (0-255). Hexadecimal colour codes do not work in Java.

2. Another way to do this is to go into any windows program, then in any colour tool, select **more colors**, and the **custom** tab. Then choose the colour you want, and copy down the red, green, and blue values shown.



Semi Transparent Colours

If you create a new colour with a fourth 0-255 value, it will be used as the "alpha" value, which controls how transparent the colour is. When you draw in a partially transparent colour, the shapes and colours of what's underneath will show through. Try it, it's cool!

Example: **gc.setColor(new Color(128, 138, 255, 127));**

The 127 is the transparency. A value of 127 or 128 is 50%.

A value of 0 is completely transparent (so you won't see anything that you draw).
A value of 255 is completely opaque, as if you had no transparency at all.

★ There are some other colour methods that you can use. Google "java color class" to see them.
Example: **gc.setColor(Color.RED.darker().darker());**

★★ You can also use HTML hexadecimal colour codes:
gc.setColor(Color.decode("#F3AA7B"));

Fonts in Java

Fonts are described by Font objects created by the Font class.

► You must **import java.awt.Font;** in order to use Font in Java. ◄

There are three parameters that you must supply (in this order) when you create a font:|

font name font style font size

Font Name:

- This is the name of a font on your computer.
- It has to be spelled exactly correctly, with spaces if need be “Times New Roman”
- It is not case sensitive so “arial” and “arIAL” both work.
- If the font is not found then it reverts to the default font.
- There are keywords for generic fonts that can be used:
  “Serif” “SansSerif” and “monospaced”

Font Style:

These are special constants in the font class.

Font.PLAIN - just plain text
Font.BOLD - bold text
Font.ITALIC - italic text
*You can add Font.BOLD +Font.ITALIC if you want both **bold and italic**.*

Font Size:

Any positive integer.

. . . . . .

► **Creating a font:**

```
Font myFont = new Font("Comic Sans MS", Font.PLAIN, 18);
```

★ TIP: creating Fonts is much slower than creating colours. Do not create fonts inside a game loop.

► **Setting and using the font:**

★ When you set the font, it uses this font for any subsequent writing that you do.

```
gc.setFont( myFont );
gc.drawString("Wazzup?", 50,50);
```

★ TIP: You can combine both steps *(and remember to turn antialiasing on as well)*:

```
gc.setFont( new Font( "Arial", Font.BOLD, 18) );
```