
Drawing Shapes and Text

Chapter 2 – Lecture Slides

Applet Example, using Eclipse

- To create our first applet, start Eclipse
- The “Workspace Launcher” appears, should just click “OK”
(click checkbox so it doesn’t appear next time).
- Create a new Project
(File >> New >> Project)
 - “Java Project” should be highlighted,
then click “Next”
 - Enter a project name, (e.g., HW1 or Lab1)
 - Click “Finish”
 - Your package should appear on the left side in the “Package Explorer”

Applet Example, using Eclipse

- Make sure the Project is highlighted, then create a new class (File >> New >> Class)
 - Verify that the “Source Folder” should list your project name.
 - In the “Name” field, enter a name for your class. For this exercise, we will name it “HelloWorld”.
 - Click “Finish”
- The text window on the right should show a class file setup for you to work on

(if it doesn't appear, double-click on the HelloWorld.java file in the Package Explorer)

Applet Example, using Eclipse

- Modify the program to appear as follows, changing the comments to have your information instead:

```
/** @Author: E. S. Boese      e-id: sugar
 * Date: Spring 2005 Course: CS150 */

import javax.swing.*;      // swing package
import java.awt.*;         // graphics package

public class HelloWorld extends JApplet
{
    public void paint( Graphics g )
    {
        g.drawString( "Hello World!", 0, 12 );
    }
}
```

Applet Example, Using Eclipse

- To run it:
 - Select
Run >> Run As >> Java Applet
 - A new window should pop-up with the Applet output displayed inside
 - If you get errors, they will appear in the Console window at the bottom of Eclipse

Drawing in the paint method

```
import java.awt.*;           // access the Graphics object
import javax.swing.*;       // access to JApplet
```

```
public class DrawEx extends JApplet
{
    public void paint( Graphics g )
    {
        // put your code here!
    }
}
```

Graphics

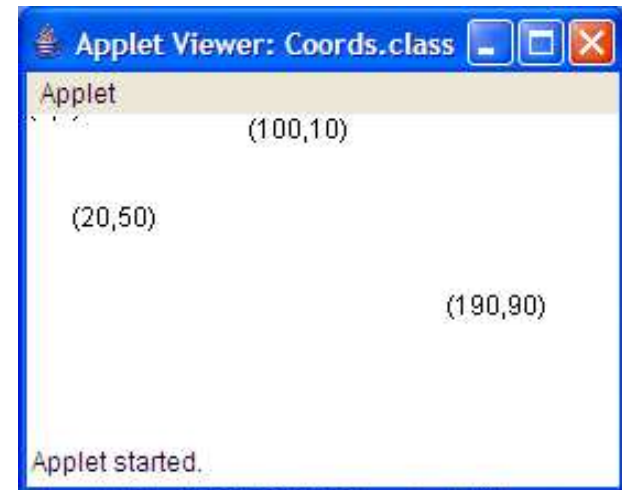
- **paint method:**
 - Basic method for showing graphical objects
 - Paint gets called automatically by browser when screen graphics shown
 - Base class has a paint method which does nothing
 - You can override the paint method by writing your own paint method
- Browser decides when to call paint method
 -
 -
 -
- We can explicitly call it

Coordinates

- (0,0) in upper left corner
- Example

```
import java.awt.*;
import javax.swing.*;
public class Coords extends JApplet
{
    public void paint( Graphics g )
    {
        g.drawString( "(0,0)",    0, 0 );
        g.drawString( "(100,10)", 100, 10 );
        g.drawString( "(20,50)",  20, 50 );
        g.drawString( "(190,90)", 190, 90 );
    }
}
```

- Why can't we see "(0,0)" printed?

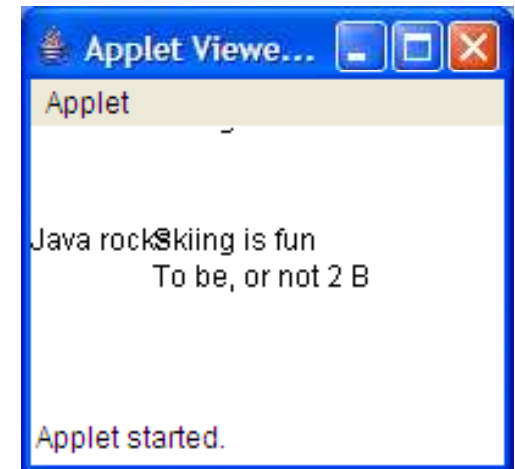


Text

- To draw text on the applet, we call

```
GraphicsObj.drawString( "text string", x, y )
```

```
import java.awt.*;           // access the Graphics object
import javax.swing.*;       // access to JApplet
public class Text1 extends JApplet
{
    public void paint ( Graphics gr )
    {
        gr.drawString ( "Hello Worldling", 0, 0 );
        gr.drawString ( "Java rocks", 0, 50 );
        gr.drawString ( "Skiing is fun", 50, 50 );
        gr.drawString( "To be, or not 2 B", 50, 65 );
    }
}
```



Graphics g

The name for the Graphics object in the parameter must match the name of the object when we call drawString

```
public void paint( Graphics grp )  
{  
    grp.drawString( "Some text to display", x, y );  
}
```

```
public void paint( Graphics g )  
{  
    g.drawString( "Some text to display", x, y );  
}
```

Font

- Don't reference special fonts that most people don't have installed on their systems! (e.g. Cursive, Alien, etc.)
 - Standard font names:
 - Serif, SanSerif, Monospaced, Dialog (Dialog is default font)
- ```
Font f = new Font("Serif", Font.PLAIN, 14);
g.setFont(f);
```
- Font.BOLD, Font.ITALIC, Font.PLAIN
    - How would you do bold and italic?



# Font Example

```
import java.awt.*;
import javax.swing.*;
public class TextFonts extends JApplet
{
 public void paint (Graphics g)
 {
 g.drawString ("Hello World", 0,10);
 Font small = new Font("Serif", Font.PLAIN, 8);
 g.setFont(small);
 g.drawString ("Java rocks", 0, 50);
 Font big = new Font("SanSerif", Font.BOLD + Font.ITALIC, 36);
 g.setFont(big);
 g.drawString ("Skiing is fun", 50, 50);
 }
}
```



# Colors

- Java Method:

  - `.setColor( color )`

- color should be

  - `Color.name`

where name can be:

- **BLACK,**                      **BLUE,**                      **CYAN,**    **YELLOW,**  
    **DARKGRAY,**              **GRAY,**                      **GREEN,**  
    **LIGHTGRAY,**              **MAGENTA,**              **ORANGE,**  
    **PINK,**                      **RED,**                      **WHITE**

- Example:



# Colors

- Make your own color

```
Color mine = new Color(0, 255, 128);
```

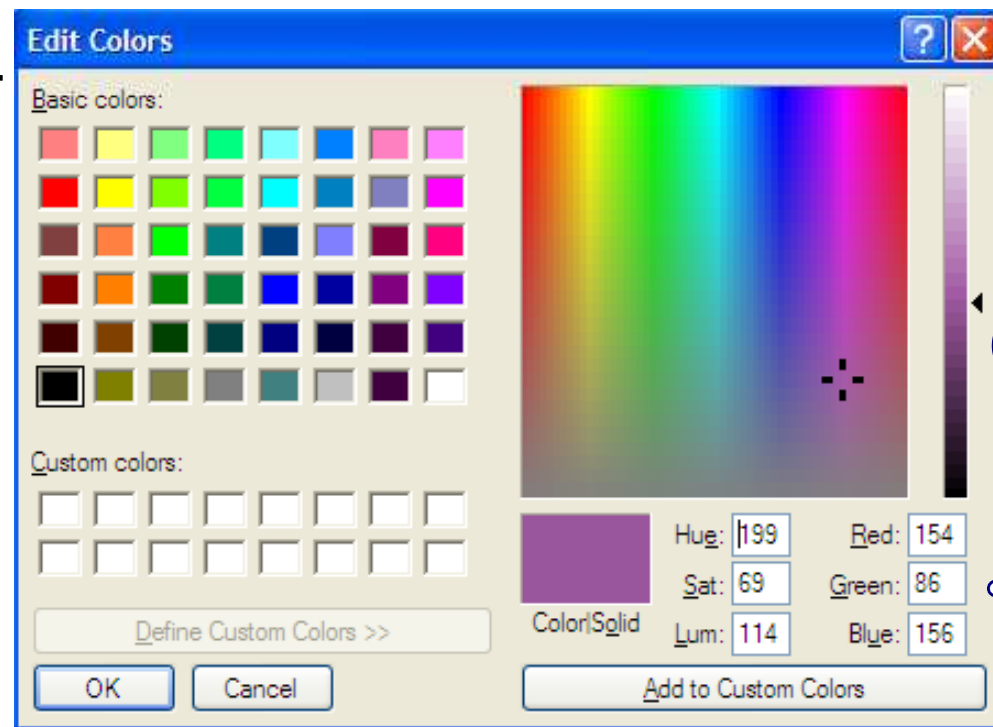
- values are the amount of **red**, **green**, and **blue** (RGB values)
- values are between 0 and 255, where 0 is black and 255 is white
- Example:
  
- Use your paint program to determine values

- Java statement example:

```
setForeground(new Color(33, 200, 190));
```

# Custom Colors

- Websites and paint programs can help determine the RGB values for a color
- Example: Microsoft Paint
  - Colors
  - Edit Colors
  - Define Custom Colors...



RGB values as displayed inside MS Paint

# Color Example

```
import java.awt.*;
import javax.swing.*;
public class ColorEx extends JApplet
{
 public void paint (Graphics g)
 {
 g.setColor(Color.RED);
 g.drawString("My favorite color", 30, 45);
 g.setColor(new Color(12,34,52));
 g.drawString("Can't wait to ski again", 30, 53);
 }
}
```



# Drawing

- draw – draws an outline of the shape
- fill – fills in the shape in the current color
- draw3D

`drawRect( x, y, w, h )`

`fillRect( x, y, w, h )`

`draw3DRect( x, y, w, h )`

`fill3DRect( x, y, w, h )`

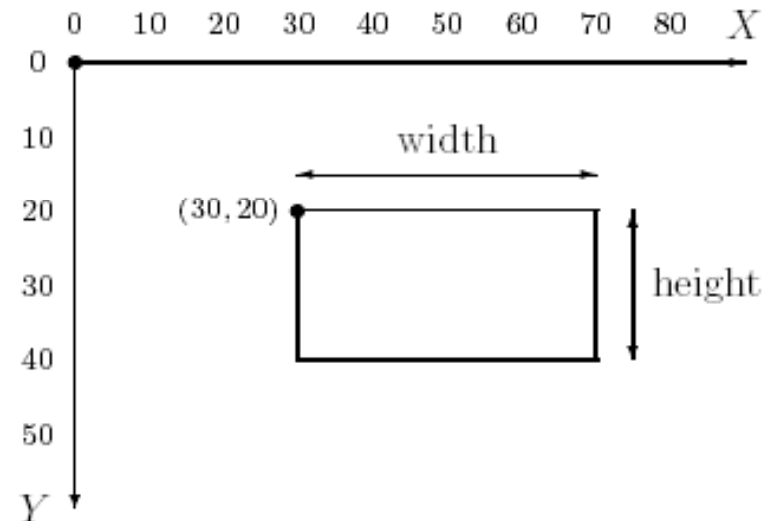
`drawLine( x, y, x2, y2 )`

`drawOval( x, y, w, h )`

`fillOval( x, y, w, h )`

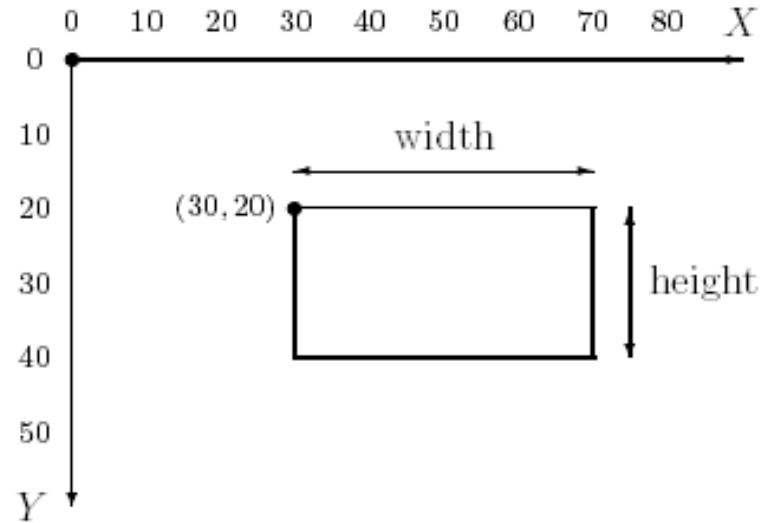
`drawArc( x, y, w, h, stA, arcA)`

`fillArc( x, y, w, h, stA, arcA)`



# Drawing

- draw
- fill
- draw3D



```
public void drawRect(int x, int y, int w, int h)
```

```
public void fillRect(int x, int y, int w, int h)
```

```
public void draw3DRect(int x, int y, int w, int h)
```

```
public void fill3DRect(int x, int y, int w, int h)
```

```
public void drawLine(int x, int y, int x2, int y2)
```

```
public void drawOval(int x, int y, int w, int h)
```

```
public void fillOval(int x, int y, int w, int h)
```

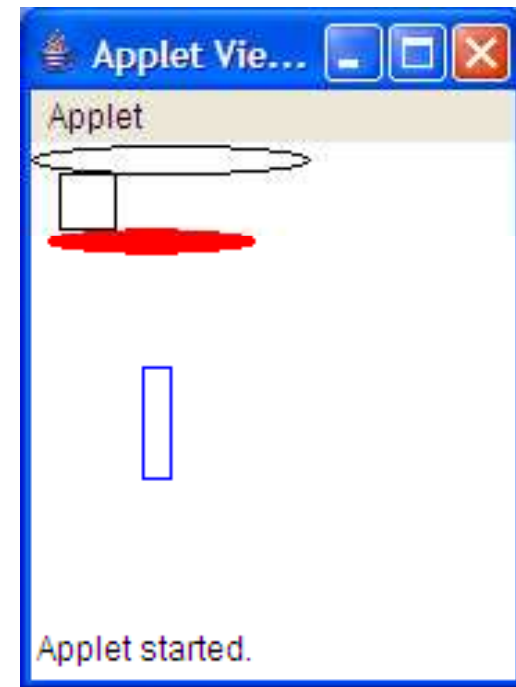
```
public void drawArc(int x, int y, int w, int h, int stA, int arcA)
```

```
public void fillArc(int x, int y, int w, int h, int stA, int arcA)
```

# Drawing Example

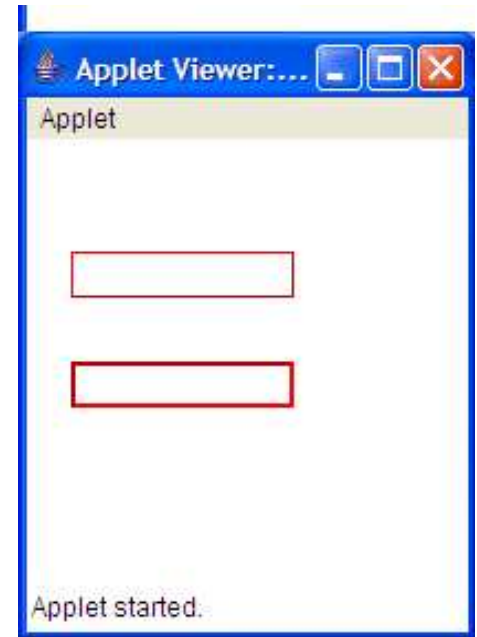
```
import java.awt.*; // access the Graphics object
import javax.swing.*; // access to JApplet
```

```
public class Shapes extends JApplet
{
 public void paint (Graphics g)
 {
 g.drawOval(0,0, 100, 10);
 g.drawRect (10,10, 20, 20);
 g.setColor(Color.BLUE);
 g.drawRect (40,80, 10, 40);
 g.setColor(Color.RED);
 g.fillOval (5,30, 77, 10);
 }
}
```



# 3DRect Example

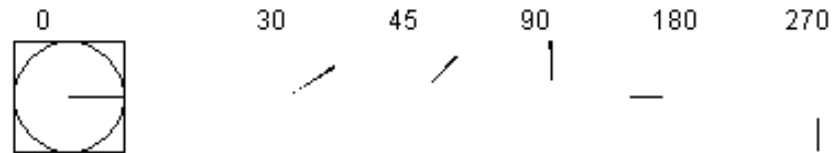
```
import java.awt.*;
import java.swing.*;
public class Rect3D extends JApplet
{
 public void paint(Graphics g)
 {
 // raised
 g.setColor(Color.RED);
 g.draw3DRect(20,50, 100, 20, true);
 g.draw3DRect(21,51, 98, 18, true);
 // sunken
 g.draw3DRect(20,200, 100, 20, false);
 g.draw3DRect(21,201, 98, 18, false);
 }
}
```



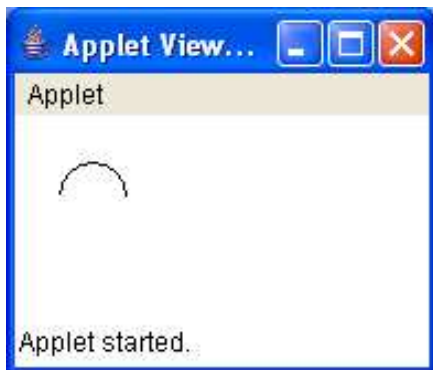
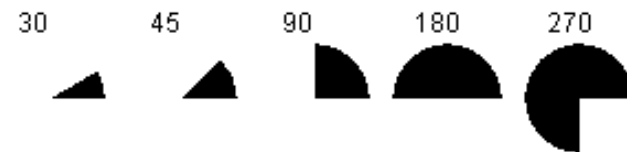
# Arc

```
public void drawArc(int x, int y, int w, int h, int stA, int arcA)
public void fillArc(int x, int y, int w, int h, int stA, int arcA)
```

- **stA = starting angle:**



- **arcA = arch angle**



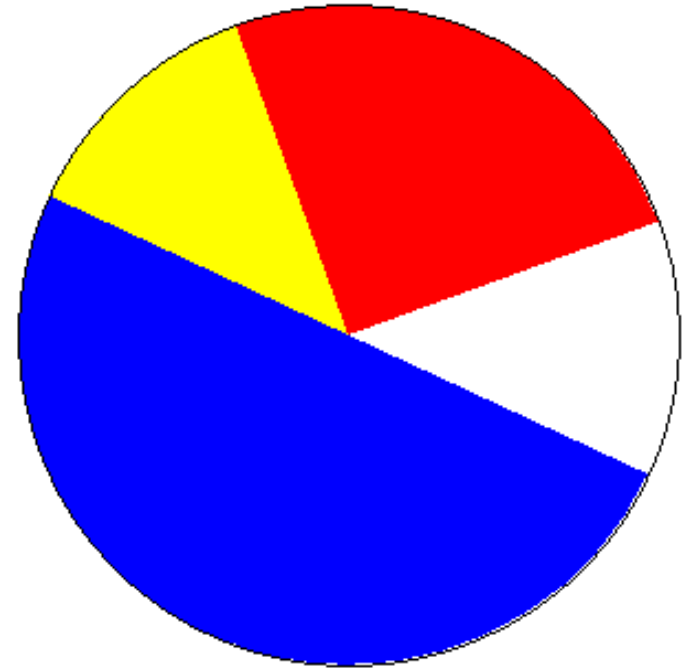
Example:

```
g.drawArc(20,20, 30, 30, 0, 180);
```

What should the above example look like?

# Arc Example

```
import javax.swing.*;
import java.awt.*;
public class PieChart extends JApplet
{
 public void paint(Graphics g)
 {
 // pie
 g.setColor(Color.red);
 g.fillArc(20,20, 300, 300, 20, 90);
 g.setColor(Color.yellow);
 g.fillArc(20,20, 300,300, 110,45);
 g.setColor(Color.blue);
 g.fillArc(20,20, 300,300, 155, 180);
 // outline
 g.setColor(Color.black);
 g.drawArc(20,20, 300, 300, 0, 360);
 }
}
```



Drawing the outline has  
to be done last – Why?

# Polygon

- Add as many points we want
- Order is important – like connect-the-dots

```
Polygon poly;
poly = new Polygon();

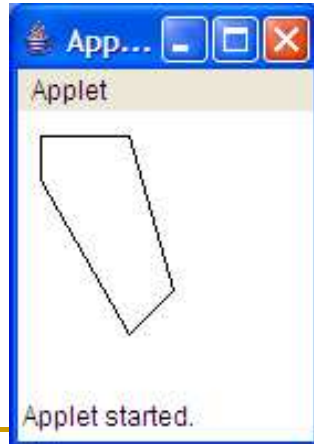
poly.addPoint(x, y);
```

- x and y are our x and y coordinates for the point.
- Draw a polygon:

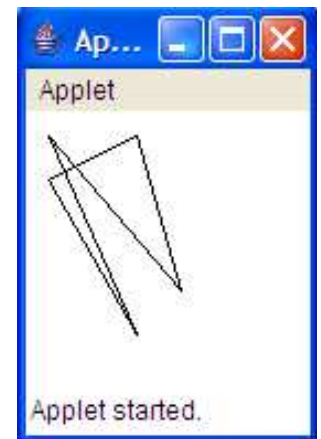
```
g.drawPolygon(poly);
g.fillPolygon(poly);
```

# Polygon

```
import javax.swing.*;
import java.awt.*;
public class PolyEx1 extends JApplet
{
 public void paint(Graphics g)
 {
 Polygon pg = new Polygon();
 pg.addPoint(10, 10);
 pg.addPoint(50, 10);
 pg.addPoint(70, 80);
 pg.addPoint(50, 100);
 pg.addPoint(10, 30);
 g.drawPolygon(pg);
 }
}
```



```
import javax.swing.*;
import java.awt.*;
public class PolyEx2 extends JApplet
{
 public void paint(Graphics g)
 {
 Polygon pg = new Polygon();
 pg.addPoint(10, 10);
 pg.addPoint(50, 100);
 pg.addPoint(10, 30);
 pg.addPoint(50, 10);
 pg.addPoint(70, 80);
 g.drawPolygon(pg);
 }
}
```



# Images

- Call method **getImage**
- Graphics method **drawImage**  
*Needs the keyword **this***

```
g.drawImage(imageVariable, x, y, this);
 // scale the image: d:destination, s:source image
g.drawImage(img, xd1, yd1, xd2, yd2, xs1, ys1, xs2, ys2, this);
```

- `getCodeBase( )` determines the location of your files:
  - Eg. Windows: C:\Documents and Settings\username\Desktop\java
  - E.g. Linux: \usr\home\username\java
  - Eg. Internet: <http://www.cs.colostate.edu/~username/java>

# Image Example

```
import javax.swing.*;
import java.awt.*;
public class ImageEx extends JApplet
{
 public void paint(Graphics g)
 {
 Image img = getImage(getCodeBase(), "Lion.jpg");
 g.drawImage(img, 0,0, this);
 }
}
```



---

# Image format types

- Web supports 3 major formats:
  - - Joint Photographic Experts Group
    - Supports 24-bit colors ( $2^{24}$  number of colors)
    - Best on photographs, naturalistic artwork types
    - NOT good on lettering, simple cartoons, B/W line drawings
    - Compression causes loss of detail
  - - Graphics Interchange Format
    - 8-bit color (supports at most 256 colors)
    - Gif89a: supports transparency, animation and interlacing
  - - Portable Network Graphics
    - Free compression algorithm since gif became patented
    - Better compression than .gif
    - Supports millions of colors

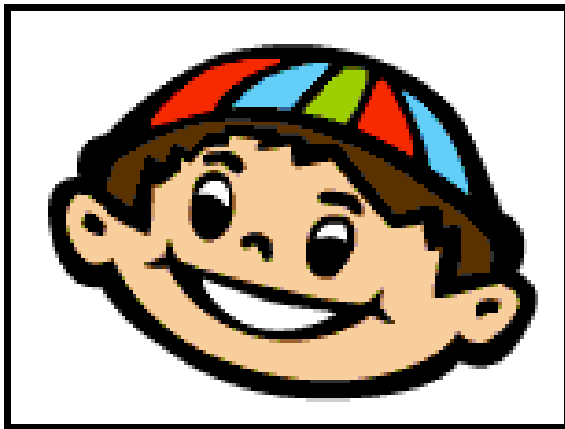
---

# Image types – Why care?

- Can drastically change size of the file
  - Increase loading speed
  - Decrease space used to store image on web server
- Desirable file sizes
  - 50-300k approx
- Decreasing the file size of an image
  - Shrink image size
  - Use different compression (gif/jpg/png)s

# .jpg vs .gif

Both of these images are less than 2.42K

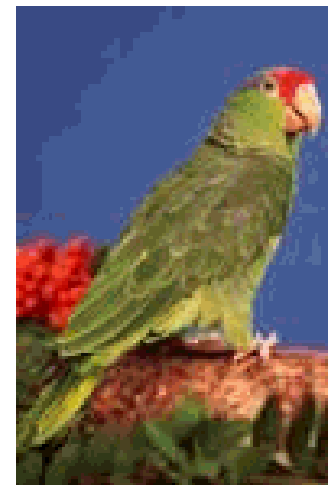


**GIF** (2,440 bytes)

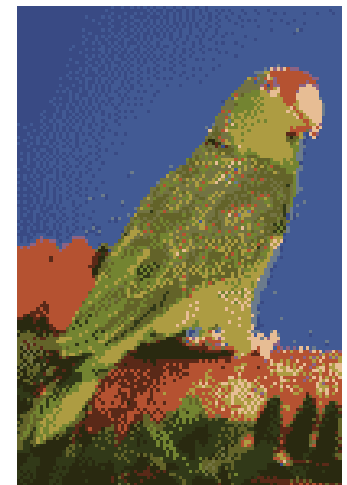


**JPEG** (2,469 bytes)

Both of these images are less than 3.88K



**JPEG** (3,849 bytes)



**GIF** (3,968 bytes)

from [builder.com.com](http://builder.com.com)

# jpeg compression experiment

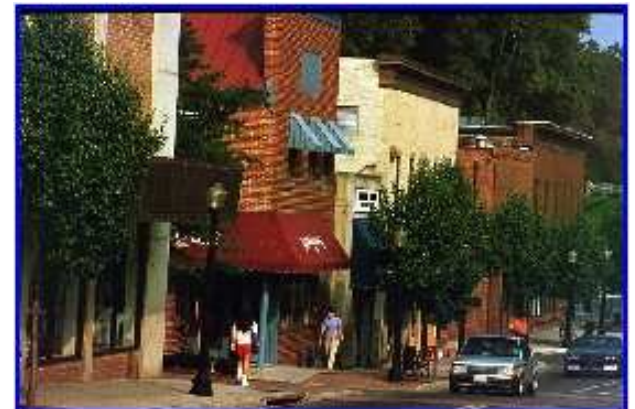
<http://www.cs.sfu.ca/CourseCentral/365/li/material/cgi-bin/whichjpeg.cgi>



Factor: 5 Size=9438 bytes



Factor: 25 Size=29360 bytes



Factor: 100 Size=326321 bytes

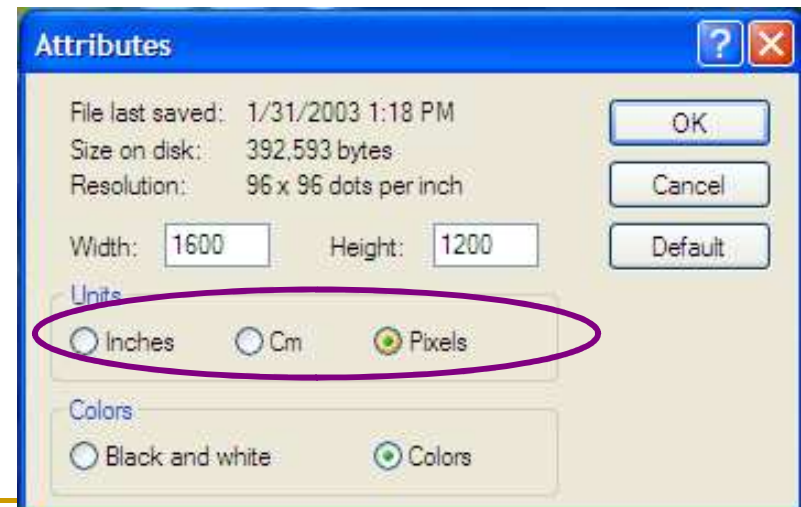
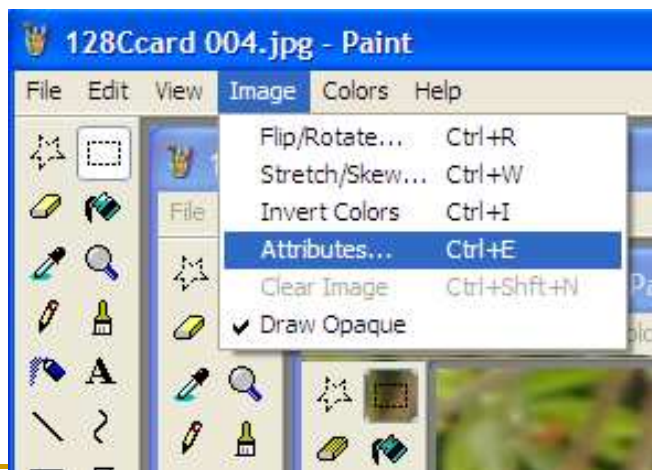
---

# Modifying Images - Programs

- Microsoft Paint
- Photoshop
- PaintShopPro
- Gimp
- Programs that came with your digital camera and/or printer

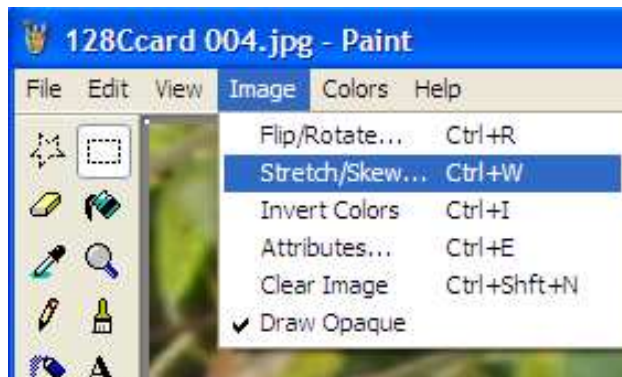
# Image size

- Images on the Web/applet should fit within the screen or area where you want it.
- Digital cameras and photos shops usually return images to you at 1600x1200 – really large!
- To see the size of your image in MS Paint, Click on **Image** then **Attributes**



# Image size (con't)

- 640x400 pixels is probably the biggest size you'd want, or even smaller
- To resize an image in MS Paint, Click on **Image** then **Stretch/Skew...**
- Change the **Stretch** only, and keep the % for width and height the same This keeps the image proportionally correct
- For a 1600x1200 photo, go to either 30% or 20%



---

# Image size (con't)

- Terminology

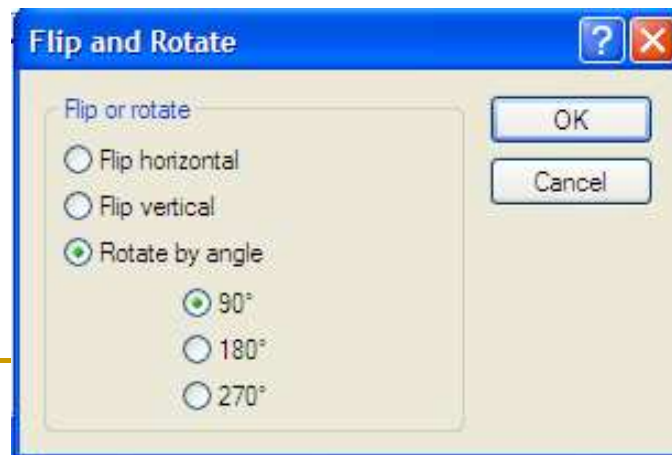
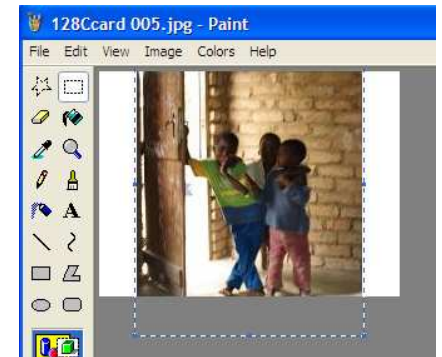


- Java

- There are ways to scale and modify images within java, but gets very complex
  - Easier to modify image in a graphics program instead

# Image Orientation

- Some photos from digital cameras/photo shops come back sideways
- Rotate in MS Paint using **Image** then **Flip/Rotate...**  
**Rotate by angle...**  
select either **90** to rotate right or **270** to rotate left



# Pixels



- Pixels are small rectangular areas on the screen that hold a solid color.
- More pixels means better quality of the image – less ragged



---

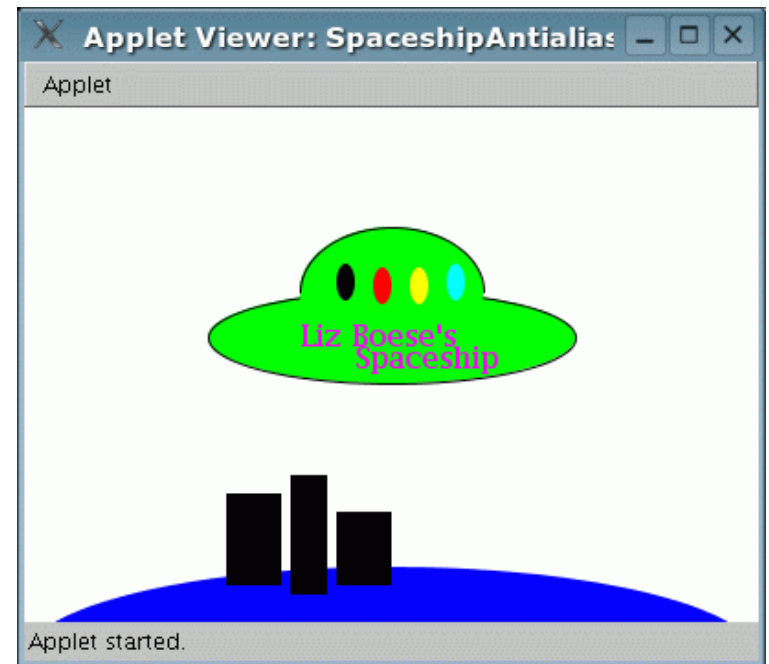
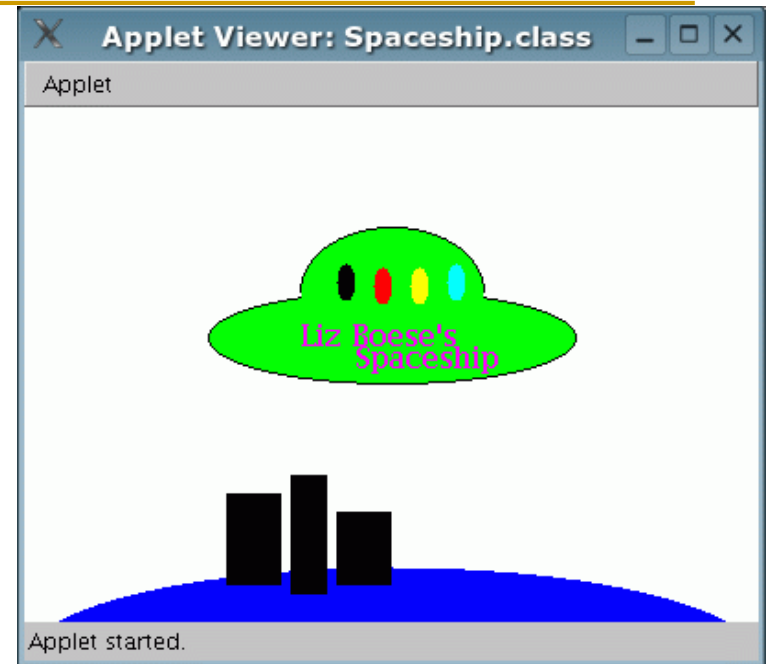
# Transparency in images

- MS Paint doesn't do it
- use Photoshop, PaintShopPro or other Graphics tool
- On-line ones:  
<http://stuff.mit.edu/tweb/map.html>

# Anti-Aliasing

- Anti-aliasing helps **smooth** the lines.
- Works on both graphics and text
- Can slow-down drawing process
- To use anti-aliasing, need to convert Graphics object to Graphics2D

```
public void paint(Graphics grph)
{
 Graphics2D g2d = (Graphics2d)grph;
 g2d.setRenderingHint(
 RenderingHints.KEY_ANTIALIASING,
 RenderingHints.VALUE_ANTIALIAS_ON);
 // now draw your code with g2d.drawRect, etc.
}
```



---

# Funky Tricks:

- Groovy text, with animation
  - [www.3dtextmaker.com](http://www.3dtextmaker.com)
- Changing color indexes
  - <http://builder.com.com/5102-6371-5047689.html>

---

# Lecture Exercise

- Let's draw a smiley face!

---

# Summary

- paint method
- Coordinate system
- Text and fonts
- Color
- Drawing shapes
  - line
  - circles and ovals
  - squares and rectangles
  - arcs
  - polygons
  - images
- Java supported image file types
- Differences between image types
- Changing image size
- Transparency
- Anti-aliasing