



Course Name:
Advanced Java



Lecture 21

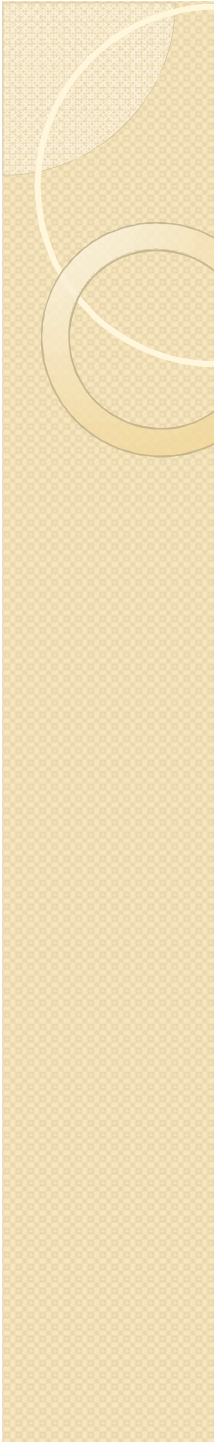
Topics to be covered

- Coordinate Transformations
- Clipping
- Transparency and Composition
- Rendering Hints
- Readers and Writers for Images, Image Manipulation
- Printing, The Clipboard, Drag and Drop

Coordinate Transformation

There are Four transformations:

- **Scaling** – blowing up or shrinking ,all distances from a fixed point. The scale method of the Graphics 2D class sets the coordinate transformation of the graphics context to a scaling transformation. That transformation changes user coordinates (user-specified units) to device coordinates (pixels).
- **Rotation** – rotating all points around a fixed center.

- 
- **Translation** – moving all points by a fixed amount.
 - **Shear** – leaving one line fixed & “sliding” the lines parallel to it by an amount that is proportional to the distance from the fixed line.
 - You supply both a rotation & scaling transformation.

```
g2.rotate(angle);  
g2.scale(2, 2);  
g2.draw();
```

Clipping

By setting a clipping shape in the graphics context, you constraint all drawing operations to the interior of that clipping shape.

```
g2.setClip(clipShape) ;  
g2.draw(shape) ;
```

However, in practice you don't want to call the setClip operation, since it replaces any existing clipping shape that the graphics context may have. Instead call the clip method.

```
g2.clip(clipShape) ;
```



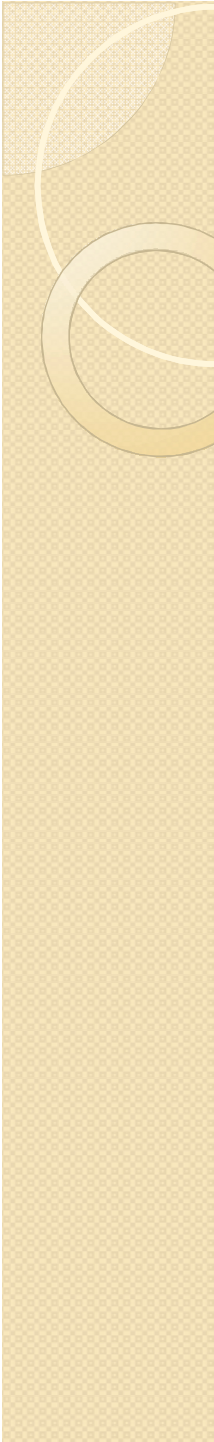
The clip method intersects the existing clipping shape with the new one that you supply.

If you just want to apply a clipping area temporarily, then you should first get the old clip, then add your new clip, & finally restore the old clip when you are done :

```
Shape oldClip = g2.getClip() ;    // save old clip
g2.clip(clipShape) ;              // apply temporarily
clip
gr.setClip(oldClip) ;             //restore old clip
```

Transparency & Composition

- Porter & Duff two researchers in the field of computer graphics, have formulated 12 possible composition rules for this blending process. The java 2D API implements all of these rules. These rules are:
 - 1) **CLEAR** – source clears destination.
 - 2) **SRC** – source overwrites destination & empty pixels.
 - 3) **DST** – Source does not affect destination.
 - 4) **SRC_OVER** – Source blends with destination & overwrites empty pixels.
 - 5) **DST_OVER** – Source does not affect destination & overwrites empty pixels.

- 
- 6) **SRC_IN** – Source overwrites destination.
 - 7) **SRC_OUT** – Source clears destination & overwrites empty pixels.
 - 8) **DST_IN** – Source alpha modifies destination.
 - 9) **DST_OUT** – Source alpha complement modifies destination.
 - 10) **SRC_ATOP** – Source blends with destination.
 - 11) **DST_ATOP** – Source alpha modifies destination. Source overwrites empty pixels.
 - 12) **XOR** – Source alpha complement modifies destination. Source overwrites empty pixels.

Rendering Hints

Antialiasing – This technique removes the “jaggies” from slanted lines & curves.

A slanted line must be drawn as a “staircase” of pixels. Rather than drawing each pixel completely on or off. You can color in the pixels that are partially covered, with the color value proportional to the area of the pixel that the line covers, and then the result looks much smoother.

Antialiasing takes a bit longer because it takes time to compute all those color values.

You can request the antialiasing like :

```
g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
```

Example

For Ex:

```
RenderingHints hints =new RenderingHints();
```

```
hints.put(RenderingHints.KEY_ANTIALIASING,  
          RenderingHints.VALUE_ANTIALIAS_ON);
```

```
hints.put(RenderingHints.KEY_TEXT_ANTIA  
          LIASING,RenderingHints.VALUE_TEXT_ANT  
          IALIAS_ON);
```

```
g2.setRenderingHints(hints);
```

Readers for Images

To load an image, use the static read method of the ImageIO class:

```
File f = . . .;
```

```
BufferedImage image = ImageIO.read(f);
```

The ImageIO class picks an appropriate reader, based on the file type. It may consult the file extension and the "magic number" at the beginning of the file for that purpose. If no suitable reader can be found or the reader can't decode the file contents, then the read method returns null.

Writers for Images

Writing an image to a file is just as simple:

```
File f = . . . ;
```

```
String format = . . . ;
```

```
ImageIO.write(image, format, f);
```

Here the format string is a string identifying the image format, such as "JPEG" or "PNG".

The ImageIO class picks an appropriate writer and saves the file.

Obtaining Readers and Writers for Image File Types

- For more advanced image reading and writing operations that go beyond the static read and write methods of the ImageIO class, you first need to get the appropriate ImageReader and ImageWriter objects.
- The ImageIO class enumerates readers and writers that match one of the following:
 - an image format (such as "JPEG")
 - a file suffix (such as ".jpg")
 - a MIME type (such as "image/jpeg")

Example

For example, you can obtain a reader that reads JPEG files as follows:

```
ImageReader reader = null;
```

```
Iterator iter =
```

```
    ImageIO.getImageReadersByFormatName("JPEG");
```

```
if (iter.hasNext) reader =  
    (ImageReader)iter.next();
```

Image Manipulation Printing

Graphics Printing –

To print a 2D graphic, the graphic can contain text in various fonts .To generate a printout, u take care of these two tasks:

- a) Supply an object that implements the Printable interface.
- b) Start a print job.

The printable interface has a single method:

```
int print(Graphics g,PageFormat format, int page)
```

This method is called whenever the print engine needs to have a page formatted for printing. Your code draws the text & image that are to be printed onto the graphics context. The page format tells u the paper size & the print margins. The page number tells u which page to render.



To start a print job, u use the PrinterJob class. Firstly u call the `getPrinterJob` method to get a print job object. Then set the `Printable` object that u want to print.

```
Printable canvas=.....;  
PrinterJob job = PrinterJob.getPrinterJob();
```

```
Job.setPrintable(canvas);
```

Before starting the print job, u should call the `printDialog` method to display a print dialog box. That dialog box gives the user a chance to select the printer to be used(in case multiple printers are available), the page range that should be printed & various printer settings.