1 Notes on Graphics.drawImage(...)

There are 6 forms:
(1) Graphics.drawImage(Image, int, int, ImageObserver)
(2) Graphics.drawImage(Image, int, int, Color, ImageObserver)
(3) Graphics.drawImage(Image, int, int, int, int, ImageObserver)
(4) Graphics.drawImage(Image, int, int, int, int, Color, ImageObserver)
(5) Graphics.drawImage(Image, int, int, int, int, int, int, int, int, ImageObserver)
(6) Graphics.drawImage(Image, int, int, int, int, int, int, int, int, Color, ImageObserver)

Explaination of the arguments:

- In each case, you need an Image to draw, and an Image Observer.
- If you give a Color argument, it replaces any transparent bits in the image – the effect is as if the image is painted over a background with the transparent bits.
- Depending on whether we have 2, 4 or 8 ints, they are interpreted differently. In every case, the first 2 int gives IN WINDOW COORDINATES, the position for the upper-left corner of the DRAWN image. [Note: the DRAWN image may be a subimage of the original image]
- In case of 4 int's, the next two ints specify the width and height of the DRAWN image. The original image will be scaled to this width and height.
- In case of 8 int's, the second pair of ints specify (in WINDOW COORDINATES) the position for the lower-right corner of the DRAWN image. The third pair specify (in IMAGE COORDINATES) the position of the upper-left corner of the DRAWN image. The fourth pair specify (in IMAGE COORDINATES) the position of the lower-right corner of the DRAWN image.

Here are some useful tips for using these commands:

- The ImageObserver can be the Any component or image can be the Image Observer. So you normally just say "this" or even "null". We give more explanation of Image Observer below.
- Above we made it clear that there are two independent coordinate systems at work here: Window Coordinates and Image coordinates. A third set of coordinates is the Screen Coordinates (your Window is placed in the Screen Coordinates). For instance, here is how we placed the current window in the center of the screen:

```
// Centering the Current Window Component:
Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
```

```
    Dimension frameSize = getSize();
    int x = (screenSize.width - frameSize.width) / 2;
    int y = (screenSize.height - frameSize.height) / 2;
    setLocation(x, y); // set location of frame in center of screen
```

To know the usable area of your Component, you need to call "getSize()" which returns "Dimension" object (i.e., width, height pair of ints) as well as "getInsets()". The latter returns an "Inset" object which can be used to get the top, bottom, left and right insets for drawing:

```
public void paint(Graphics g){
  Inset ins = getInsets();
  g.drawImage(im, ins.left, ins.top, this);
}
```

Note that the insets of a Frame includes the height of the title and menubars. [4, p.112]

To get the dimensions of your image, use Image.getWidth(ImageObserver) and Image.getHeight(ImageObserver).

- In the 8 int's case, suppose the 8 integers are (x,y, x', y', u, v, u', v'). Then we expect x¡x' and y¡y' in WINDOW COORDINATES. Normally, we also expect u¡u', v¡v' in IMAGE COORDINATES. However, if you u¿u' and/or v¿v', you get an interesting effect of reversing and inverting the image!
- For Colar, you can use the predefined colors such as Color.cyan, Color.pink, etc.

2 Asynchronous Loading of Images

ImageObserver is an interface with the imageUpdate() method. And java.awt.Component implements ImageObserver. We now explain why the arguments to drawImage(...) always get an ImageObserver.

The key fact is that images are generally slow to load, especially over a network. This means that hanging up the drawImage(...) method until the whole image is available for drawing is not an acceptable default behaviour. Instead, the default behavior is to draw as must of the image as are available, and to redraw when more bits arrive.

Here is how we achieve such behaviour: when drawImage(...) is called, it registers the ImageObserver and draws the current partial image. Henceforth, whenever a portion of the image is loaded, the image producer will invoke the imageUpdate() method of the ImageObserver.

The default behavior of the imageUpdate() method is to invoke repaint() which results in a call to paint(). Normally, the paint() method will have a call to drawImage(...).

Note that drawImage(...) returns true iff the image has been completely loaded.

There is an interface called MediaTracker that automates some of the tasks of loading images.

3 Eliminating of Flicker

An image that is repeated "repainted" will flicker. That is because repaint() will call update() as soon as possible. THe default Component.update() method will erase the background of the component and then invoke the paint() method. A simple solution is to redefine the update() method to directly call paint():

```
public void update(Graphics g)  paint(g);
```

See [Geary, p.116].

4 Other Topics

We want the ability to zoom by moving a slider.

Another useful topic is scrollpane.