Graphics & Java

CSCU9N6

Contents

- Simple Graphics
- Console Code
- A Simple Screen Manager
- Anti-Aliasing & Graphics Primitives
- Images & Animation
- Double Buffering & Page Flipping

Graphics in Java

In order to render graphics in Java, we need:

- A Window Object
  - A canvas to drawn on (e.g. some form of JFrame)
- A DisplayMode Object
  - Defines the type of display we will be drawing on
  - Includes screen resolution, bit depth and refresh rate
- A Graphics Device
  - Acts as a boundary to the graphics card
  - We can set display modes and properties via this boundary
  - If you did not have this object, you would need to write specific code for the graphics card you were using...

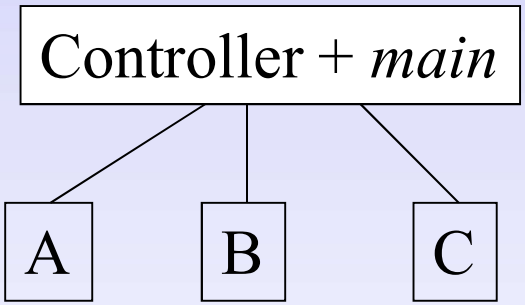
Simple Example

```
import java.awt.*;
import javax.swing.JFrame;

public class FullScreen {

    public static void main(String[] args) {

        JFrame                          window = new JFrame();
        DisplayMode          mode = new DisplayMode(1024,768,16,75);

        GraphicsEnvironment env=GraphicsEnvironment.getLocalGraphicsEnvironment();
        GraphicsDevice      dev = env.getDefaultScreenDevice();

        dev.setFullScreenWindow(window);
        dev.setDisplayMode(mode);

        try { Thread.sleep(5000); } catch (Exception e) { };

        window.dispose();
        dev.setFullScreenWindow(null);
    }
}
```

Writing Console Based Code

Console (Command Line) code has an unusual format:

- Somewhere in our code we must declare one static method called *main*
- Normally you will put this in the highest level class
- *main* will create an instance of the high level class, then call a top level method in it to get things going



Console Code - Example

```
public class Controller {

   // Main is placed inside high level class
   // although it doesn't really 'belong' to it
   public static void main(String[] args) {
      // We use main to create an instance of the high level class with relevant parameters
      Controller m = new Controller (...);
      // then call some method to start things rolling
      m.go(...);
   }

   public Controller(...) {...} // Constructor for Controller

   // go might use the internal methods skip and hop and classes Apple, Ball & Clown
   public void go(...) {
      Apple a = new Apple();
      Ball b = new Ball();
      Clown c = new Clown();

      skip(a,b);
      hop(c);
       ...
   }
   // skip and hop are private methods belonging to Controller
   private void skip(Apple a, Ball b) {...}

   private void hop(Clown c) {...}
}
```

Console Graphics Example

Now that we have looked at the general format for a console based application, we can apply it to creating a simple screen manager that will set up:

- A full screen window
- Remove borders
- Prevent the window from being resized
- Draw something to the screen

You can use this example as a starting point for a full screen application

A Simple Screen Manager - 1

```
import java.awt.*;
import javax.swing.JFrame;

/**   The SimpleScreenManager class manages initializing and  displaying full screen graphics modes. */
public class SimpleScreenManager {

   private GraphicsDevice device;

   public SimpleScreenManager() {
      GraphicsEnvironment e=GraphicsEnvironment.getLocalGraphicsEnvironment();
      // Store a reference to the graphics device we are using
      device = e.getDefaultScreenDevice();
   }

   /** Enter full screen mode and change the display mode.  */
   public void setFullScreen(DisplayMode displayMode,JFrame window)   {
      window.setUndecorated(true);
      window.setResizable(false);

      device.setFullScreenWindow(window);
      if (displayMode != null && device.isDisplayChangeSupported())
      {
         try { device.setDisplayMode(displayMode); }
         catch (IllegalArgumentException ex) {  /* ignore - illegal mode for this device */ }
      }
   }
```

A Simple Screen Manager - 2

```
import java.awt.*;
import javax.swing.JFrame;

public class FullScreenTest extends JFrame {

    public static void main(String[] args) {
        DisplayMode dm = new DisplayMode(1024,768,16,75);
        FullScreenTest test = new FullScreenTest();
        test.go(dm);
    }

    public void go(DisplayMode displayMode) {
        setBackground(Color.blue);
        setForeground(Color.white);
        setFont(new Font("Dialog", 0, 24));

        SimpleScreenManager screen = new SimpleScreenManager();
        try {
            screen.setFullScreen(displayMode, this);
            try { Thread.sleep(5000); }
            catch (InterruptedException ex) { }
        }
        finally { screen.restoreScreen(); }
    }
public void paint(Graphics g) {
                g.drawString("Hello World!", 20, 50);
                g.drawOval(100,100,30,40);
    }
}
```

Adding Anti-Aliasing

Adding Anti-Aliasing in Java is relatively simple, although there will be a processing overhead

- Note that we can apply text and graphics anti-aliasing separately

```
public void paint(Graphics g) {

    if (g instanceof Graphics2D)
    {
            Graphics2D g2D = (Graphics2D)g;
            g2D.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
                                 RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
            g2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                                 RenderingHints.VALUE_ANTIALIAS_ON);

    }
    g.drawString("Hello World!", 20, 50);
    g.drawOval(100,100,30,40);

}
```

Standard Graphics Primitives

drawString(String str,int x,int y)

- g.drawString("Hello World!", 20, 50);

drawLine(int x1, int y1, int x2, int y2)

- g.drawLine(20,80,120,80);

drawOval(int x, int y, int width, int height)

- g.drawOval(100,100,30,40);

drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)

- g.drawArc(200,100,40,40,45,135);

draw3DRect(int x, int y, int width, int height, boolean raised)

- g.draw3DRect(300,100,40,30,true);

See API guide for the class Graphics for more examples:
http://www.oracle.com/technetwork/java/api-141528.html

- Look under java.desktop for AWT (contains Graphics2D) and Swing

Drawing Images

```
public void go(DisplayMode displayMode) {

      background = new ImageIcon("images/background.jpg").getImage();
      pic = new ImageIcon("images/translucent.png").getImage();

      ....Code as before....
}

public void paint(Graphics g) {
      g.drawImage(background,0,0,null);
      g.drawImage(pic,20,20,null);
      g.drawImage(pic,120,120,null);
}
```

Sprites & Animation

Animation of an Object

- An animation can be represented as a set of image frames
- Each frame will be shown for a given time

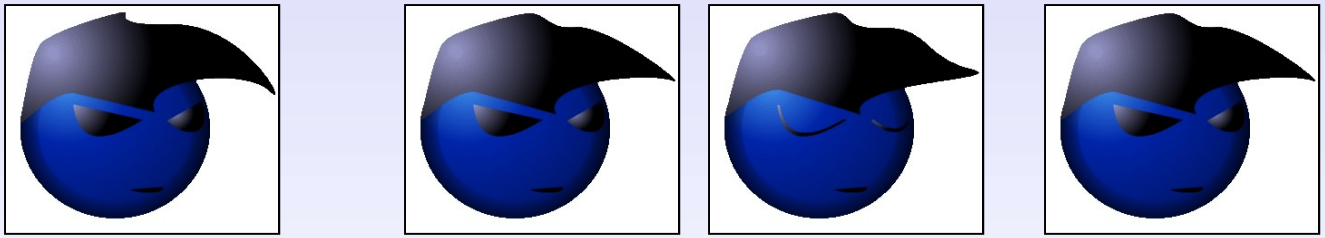
Sprites

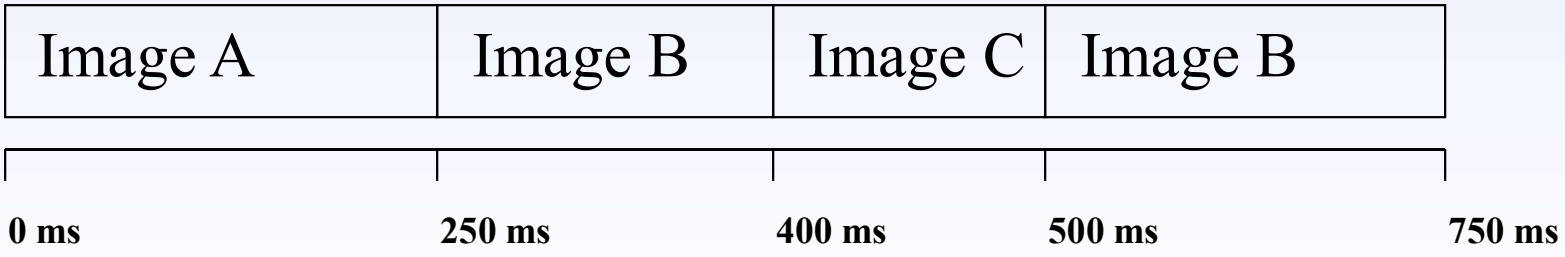
- Animated object
- Position & Velocity
- Independent action

Animation Frames

An animated object is shown as a set of frames

Each frame is shown for a given time





Animation in Java

So, how do we do this in Java?

Create an *Animation* class to control display of the relevant frame at the given time…

Animation Class contains

- A set of animation frames - *AnimFrame*
- Index of the current frame
- Current animation time
- Total time the animation takes to render
- Methods to render the animation

Animation Example - 1

```
import java.awt.Image;
import java.util.ArrayList;

/**
   The Animation class manages a series of images (frames) and
   the amount of time to display each frame.
*/
public class Animation {

   private ArrayList<AnimFrame> frames;          // The set of animation frames
   private int currFrameIndex;                   // Which frame are we currently in
   private long animTime;                        // The current time within the total animation
   private long totalDuration;                   // The total length of the animation

   public Animation() {
      frames = new ArrayList<AnimFrame>();
      totalDuration = 0;
      start();
   }
```

Animation Example - 2

```
/** Add an image to the animation */
public synchronized void addFrame(Image image, long duration)    {
   totalDuration += duration;
   frames.add(new AnimFrame(image, totalDuration));
}

public synchronized void start() {...

/** Updates this animation's current image (frame), if necessary. */
public synchronized void update(long elapsedTime) {
   if (frames.size() > 1) {
      animTime += elapsedTime;

      if (animTime >= totalDuration) {
         animTime = animTime % totalDuration;
         currFrameIndex = 0;
      }
      while (animTime > getFrame(currFrameIndex).endTime) currFrameIndex++;
   }
}
```

Animation Example - 3

```
    /** Gets this Animation's current image. Returns null if this animation has no images.  */
    public synchronized Image getImage() {
        if (frames.size() == 0)
            return null;
        else
            return getFrame(currFrameIndex).image;
    }
    private AnimFrame getFrame(int i) { return (AnimFrame)frames.get(i); }

    private class AnimFrame {
        Image image;
        long endTime;

        public AnimFrame(Image image, long endTime) {
            this.image = image;
            this.endTime = endTime;
        }
    }
}
```

Using the *Animation* class

```
public void animationLoop() {
      long startTime = System.currentTimeMillis();
      long currTime = startTime;

      while (currTime - startTime < DEMO_TIME) {
         long elapsedTime = System.currentTimeMillis() - currTime;
         currTime += elapsedTime;

         // update animation
         anim.update(elapsedTime);

         // draw to screen
         Graphics g = screen.getFullScreenWindow().getGraphics();
         g.drawImage(anim.getImage(), 0, 0, null);
         g.dispose();

         // take a nap
         try { Thread.sleep(20); }
         catch (InterruptedException ex) { }
      }
   }
```

Animation Example

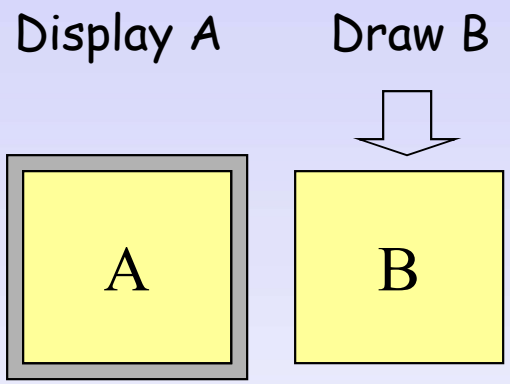
Demo

The full code for the demo, including initialisation of each animation frame will be investigated in a practical.

Flicker

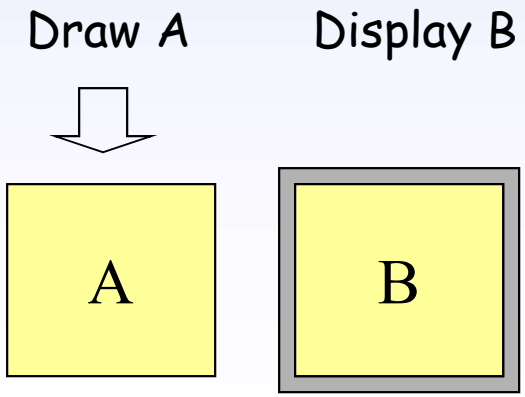
Flicker

- The images flicker due to continuous drawing of the background and then the foreground image.
- Most of the time you see the foreground image but occasionally you will see the background, giving rise to a flicker as it is then drawn over
- How can we fix this?



Double Buffering

- Render the complete image off screen to a 'buffer'
- Copy the 'buffer' to the screen in one go



Double Buffering

Double buffering involves copying the buffered image to the screen in one go

- The buffered image is the same size as the screen
- If the screen resolution is 1024 x 768 with 24 bits per pixel (3 bytes), this will need an image of 2,359,296 bytes (2.25MB) to be copied for each frame.
- 1026 x 768 x 3 = 2,359,296

Double Buffering Example

We need to use a more advanced screen manager to allow for double buffering.

- It takes care of most of the hard work
- We render our new display, then call 'update' in the animation loop to flip the buffers and show the new display
- We will look at this in more detail in a practical since there is too much code to look at here
- A similar process is followed for displaying graphics with mobile phone displays

Double Buffering - Animation Code

```
public void animationLoop() {
   long start = System.currentTimeMillis();
   long current = start;

   while (current - start < DEMO_TIME) {
      long elapsed = System.currentTimeMillis() - current;
      current += elapsed;

      // update animation
      anim.update(elapsedTime);

      // draw and update screen
      Graphics2D g = screen.getGraphics();
      draw(g);            // Draw the new frame
      screen.update();    // Now display it to the screen
      g.dispose();

     // take a nap...
   }
}
```

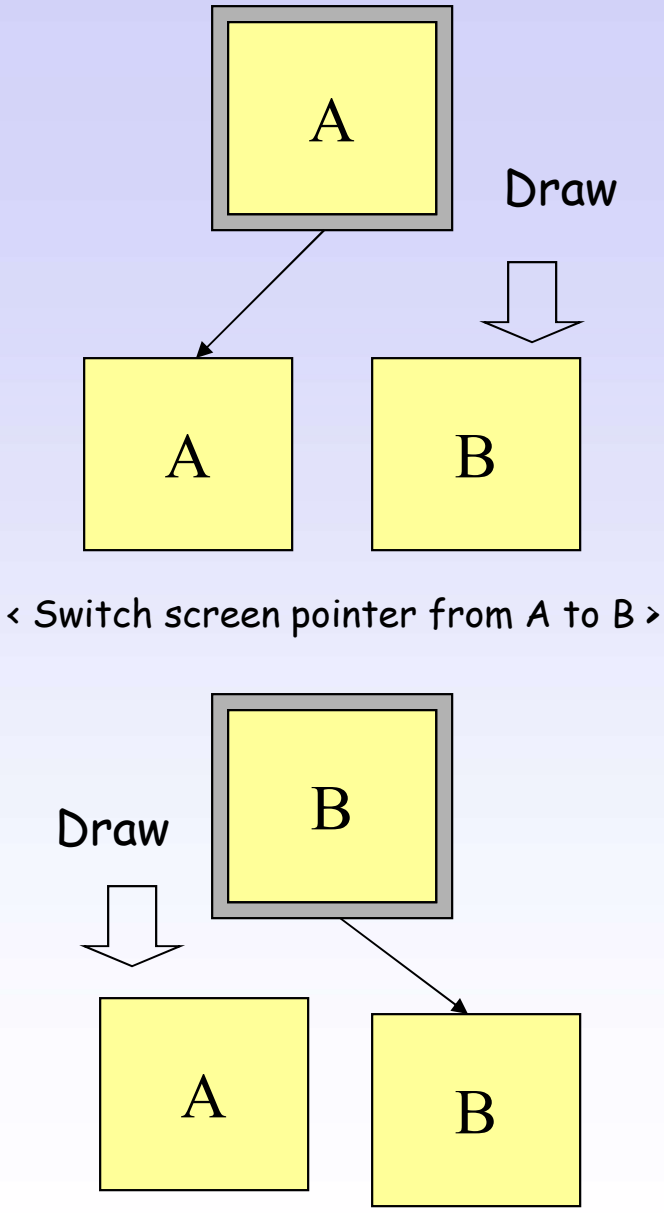
Double Buffering - Alternative

```
public void animationLoop() {
   ...
   anim.update(elapsedTime);
   if (buffer == null) // If we haven't set up the buffer yet
   {
      // Create our own buffer
      buffer = new BufferedImage(screen.getWidth(), screen.getHeight(), BufferedImage.TYPE_INT_ARGB);
      bg = (Graphics2D)buffer.createGraphics();
   }

   // draw to the buffer graphics, not the screen
   draw(bg);

   // Now get the screen graphics device
   Graphics2D g = screen.getGraphics();
   // and draw our completed image on that
   g.drawImage(buffer,null,0,0);
   g.dispose();

   // take a nap...
}
```

Page Flipping

Similar concept to Double Buffering

- Image is drawn to an off-screen buffer
- Instead of copying the entire buffer, the display is pointed to the buffer that it should use
- Program 'flips' the pointers between each rendered frame
- Executes considerably faster since no copying of large memory areas required



Tearing

- The screen is being updated with a given refresh rate (e.g. 75Hz or 75 times per second)
- What happens when a buffer is flipped (or copied) whilst the screen is being refreshed?
- Part of the screen shows the old buffer
- The rest of the screen shows the new buffer
- A 'tear' appears between the old and the new buffer
- The solution is to ensure that a buffer is flipped just before the screen is refreshed
- This is device dependent and may not be available
- See *BufferStrategy* class for example DGJ, p60

Summary

Covered

- Simple Graphics
- Console Code
- A Simple Screen Manager
- Anti-Aliasing & Graphics Primitives
- Images & Animation
- Double Buffering & Page Flipping

Next

- Sprites
- User Interface Elements