Graphics & Java

David Cairns

Contents

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 draw or paint method that will draw the actual graphics

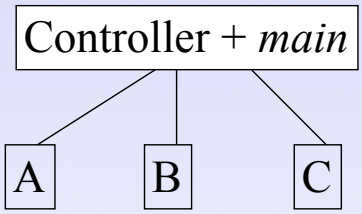
Simple Frame

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

public class SimpleFrame extends JFrame {

     public static void main(String[] args)
     {

          SimpleFrame sf = new SimpleFrame();
          sf.go();
     }

     public void go()
     {
          setSize(320,200);
          setVisible(true);
     }

    public void paint(Graphics g)
    {
          g.drawString("Hello World!", 20, 50);
          g.drawOval(100,100,30,40);
    }
}
```

Writing Console Based Code

Console (Command Line) code has an unusual format:

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



Console Code - Example

```
public class Controller {

    // Main is placed inside a 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) {...}
}
```

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

    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!", 200, 50);
    g.drawOval(200,100,30,40);
}
```


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:
https://docs.oracle.com/en/java/javase/11/docs/api/java.desktop/java/awt/Graphics2D.html

Drawing Images

```
public class SimpleFrame extends JFrame
{
      Image background;
      Image pic;

      public static void main(String[] args)
      {

            SimpleFrame sf = new SimpleFrame();
            sf.go();
      }

      public void go()
      {

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

            setSize(800,600);
            setVisible(true);
      }

      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

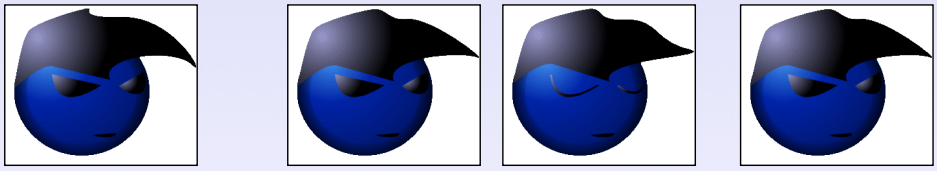
Sprites

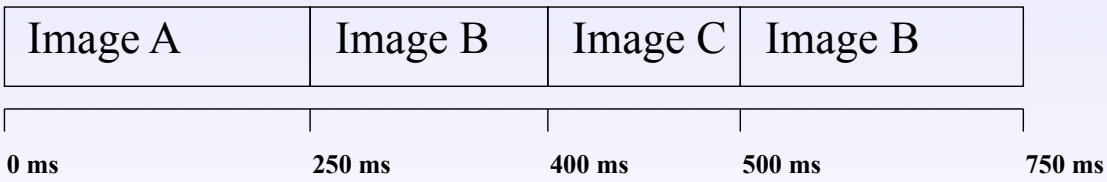
- Animated object
- Position & Velocity
- Independent action

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 Class

```
import java.awt.*;
import java.util.ArrayList;
import javax.swing.ImageIcon;

public class Animation
{
    private ArrayList<AnimFrame> frames; // The set of animation frames
    private int currFrameIndex;          // Current frame animation is on
    private long animTime;               // Current animation time
    private long totalDuration;          // Total animation time
    private float animSpeed = 1.0f;      // Animation speed, e.g. 2 will be twice as fast

    private boolean loop = true;         // True if the animation should continue looping
    private boolean looped = false;      // True if 1 animation loop has been completed
    private boolean play = true;         //   True if the animation should animate

    // Create a new, empty Animation
    public Animation()
    {
        frames = new ArrayList<AnimFrame>();
        totalDuration = 0;
        looped = false;
        start();
    }
```


Animation Class

```
    /**
     * Adds an image to the animation with the specified
     * duration (time to display the image).
     *
     * @param image      The image to add
     * @param duration   The time it should be displayed for
     */
    public void addFrame(Image image, long duration)
    {
        totalDuration += duration;
        frames.add(new AnimFrame(image, totalDuration));
    }


    /**
     *  Starts this animation over from the beginning.
     */
    public void start()
    {
        animTime = 0;
        currFrameIndex = 0;
        looped = false;
        play = true;
    }
```

Animation Class

```
// Updates this animation's current image (frame) based on how much time has elapsed.
// @param elapsedTimeTime that has elapsed since last update of the animation

public void update(long elapsedTime)
{
    if (!play) return; // If we are paused, don't update the animation

    elapsedTime = (long)(elapsedTime * animSpeed);

    if (frames.size() > 1)
    {
        animTime += elapsedTime;

        if (animTime >= totalDuration)
        {
            if (loop)
            {
               animTime = animTime % totalDuration;
               currFrameIndex = 0;
            }
            else
            {
                animTime = totalDuration;
            }
            looped = true;
        }

        while (animTime > getFrame(currFrameIndex).endTime)
            currFrameIndex++;
     }
}
```


Animation Class

```
    /**
     * Gets this Animation's current image. Returns null if this animation has no images.
     * @return The current image that should be displayed
     */
    public Image getImage()
    {
        if (frames.size() == 0)
            return null;
        else
            return getFrame(currFrameIndex).image;
    }

    public int getFrameNumber()
    {
        return currFrameIndex;
    }

    // AnimFrame - A private inner class to hold information about a given animation frame.
    private class AnimFrame
    {
        Image image;  // The image for a frame.
        long endTime; // The time at which this frame ends.

        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;
        long elapsedTime;

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

            // update animation
            anim.update(elapsedTime);

            // draw to screen
            draw(getGraphics());

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


Animation Example

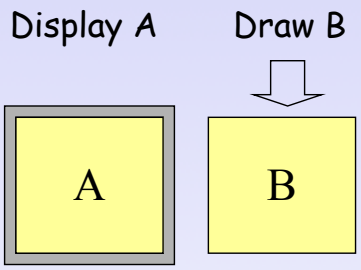
Demo

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

Flicker

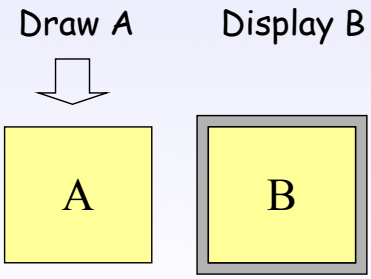
Flicker

- The images sometimes 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

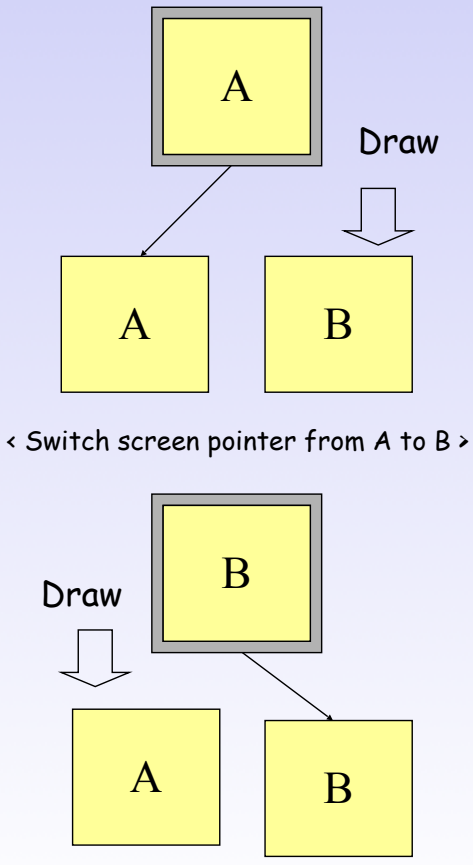
We need to use a more advanced screen manager to allow for double buffering.

- 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's too much code to look at here
- A similar process is followed for displaying graphics with mobile phone displays

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. 60Hz or 60 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
  - Graphics cards have options to only refresh after a full frame has been loaded – some also allow for this to be independent of screen refresh rate for more fluid results.
- See BufferStrategy class for example DGJ, p60

Summary

Covered

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

Next

- Sprites
- User Interface Elements