AWT Overview; Multimedia

- Abstract Window Toolkit

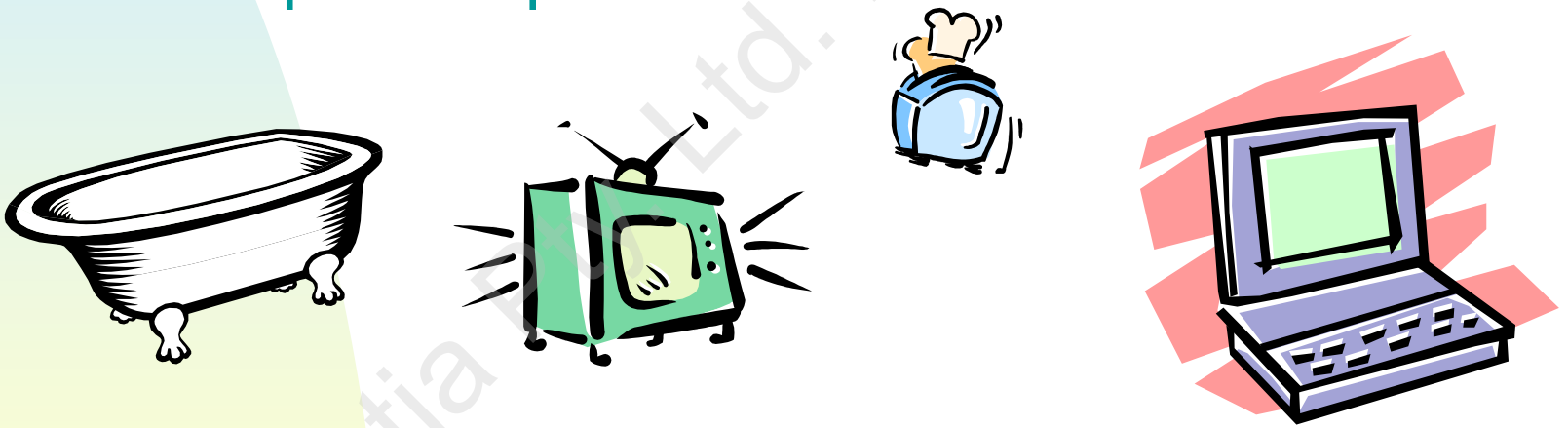
  *"Unfortunately, AWT is somewhat primitive, not very well documented, and not particularly powerful. On the other hand, it **is** platform independent."*

  "The Awkward Window Toolkit"

  "The AWT provides the means whereby a programmer can now worry about bugs in the implementation across not just one, but many different platforms!"

AWT Overview; Multimedia

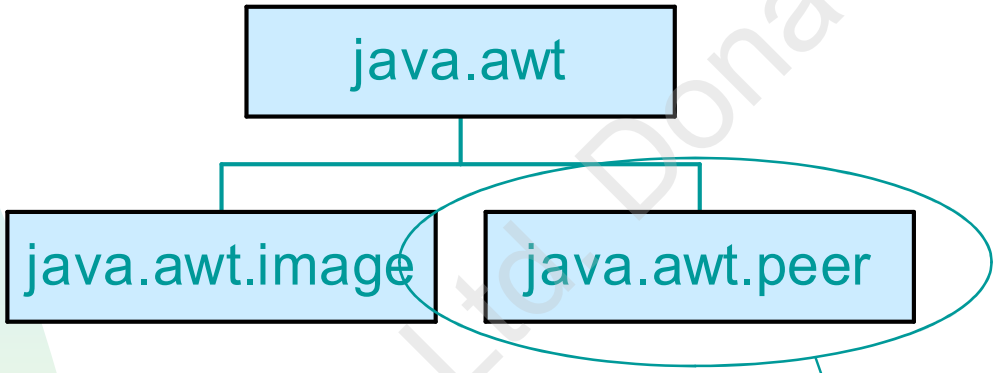
- The AWT has a grand aim: to provide the same facilities for an application embedded into a $100 PDA as for one working with the world-wide web on a 63” HDTV and another running on a supercomputer



- The (Java 1.0 version) AWT is little more than what is required to build HotJava

AWT Overview; Multimedia

- Fundamental organisation



responsible for dealing with the host OS's facilities

***very* rarely used by application programmers**

AWT Overview; Multimedia

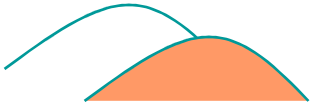
- Not just for windows—AWT handles everything that you can see/hear
- 3 aspects to java.awt class hierarchy
  - primitive graphics
  - fonts
  - standard GUI elements

AWT Overview; Multimedia

- Primitive graphics
  - shapes
  - fills
  - character strings
  - colours
- Sounds handled separately
  - java.applet.AudioClip

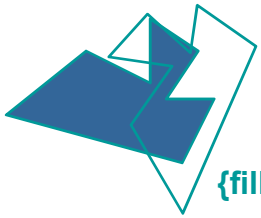
{fill, draw}Oval

{fill, draw}Rect



{fill, draw}Arc

Color



{fill, draw}Polygon

{fill, draw}3DRect

drawLine

hello world!

draw{String, chars, bytes}

{fill, draw}RoundRect

AWT Overview; Multimedia

- java.awt.Graphics class
  - must import before use
  - pretty standard drawing tools

```
public void paint (Graphics g)
  {
  g.setColor (Color.pink);
  g.fillOval (10, 10, 330, 100);

  g.setColor (Color.red);
  g.drawOval (10, 10, 330, 100);
  g.drawOval (9, 9, 332, 102);
  g.drawOval (8, 8, 334, 104);
  g.drawOval (7, 7, 336, 106);

  g.setColor (Color.black);
  g.setFont (new Font ("Helvetica", Font.BOLD, 48));
  g.drawString ("Hello World", 40, 75);
  }
```

AWT Overview; Multimedia

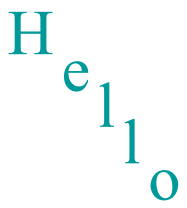
- Current colour
  - colours are 24-bit
    - constants: Orange, Cyan, Pink, Black, Blue, etc.
    - no *"α channel"* (for transparency) available
  - set{Fore, Back}ground
  - setColor
  - brighter, darker
- XOR only alternative mode
  - setXORMode (Colour)
- Standard set of cursors
  - CROSSHAIR_CURSOR, WAIT_CURSOR, etc.

AWT Overview; Multimedia

- Fonts
  - problems:
    - different platforms, different fonts
    - different fonts, different layouts
  - solutions:
    - define mappings
      - Helvetica -> adobe-helvetica / Arial
      - ZapfDingbats -> itc-zapfdingbats / WingDings
    - FontMetrics class
      - constructor FontMetrics (Font)
      - {string, char}Width
      - get{Ascent, Descent}
      - etc.

AWT Overview; Multimedia

```
public void paint (Graphics g)
  {
  String str = "Hello";
  FontMetrics fm = getFontMetrics (getFont ());
  int descent = fm.getDescent (),
      xPos = 50,
      yPos = xPos;
  for (int i = 0; i < str.length (); i ++)
    {
    char c = str.charAt (i);
    xPos += fm.charWidth (c);
    yPos += descent;
    g.drawString (c + "", xPos, yPos);
    }
  }
```



AWT Overview; Multimedia

- Update vs. paint
  - applet's context never calls paint directly. Instead calls update, which
    - clears the screen
    - then calls paint
  - can cause uncomfortable amount of flicker
    - especially with animations
  - solution:

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

AWT Overview; Multimedia

- Obtaining images
  - Applet's getImage (url {, name})
    - URL obtained from get{Code, Document}Base
      - getCodeBase: base URL for applet
      - getDocumentBase: base URL for enclosing document
    - name defines object at URL site
  - java.awt.Toolkit.getImage ({String, URL})
    - application only, of course

AWT Overview; Multimedia

- Drawing images
  - g.drawImage (image, x, y, observer)
  - image.get{Width, Height}
  - AWT assumes that an image coming across the network arrives slowly
  - first call to drawImage
    - creates thread
    - starts loading
    - registers an *ImageObserver*
  - when data is ready (observer is notified appropriately), call drawImage again to render it

AWT Overview; Multimedia

- ImageObserver interface
  - a callback that notifies of change of state
    - incremental rendering/multi-frame images, etc.
  - Component implements ImageObserver interface
    - single method: imageUpdate (img, info, x, y, w, h)
      - info: flags indicating what info about the image is available. Some combination of:
        - ERROR, ABORT
        - ALLBITS, SOMEBITS (x, y, w, h)
        - HEIGHT (h), WIDTH (w)
        - FRAMEBITS
        - PROPERTIES
  - Applet is a component, so can be a observer

AWT Overview; Multimedia

- MediaTracker
  - tracking image loading individually can get very messy
  - register them (& sounds, etc. eventually) with a MediaTracker object
    - loads images in parallel -> better apparent performance
    - can check state individually
    - can block until all loaded
  - animation
    - multithreaded example follows

AWT Overview; Multimedia

```
public class Animator extends Applet implements Runnable
  {
  private Image [] images;
  private MediaTracker tracker;
  private int num_images,
              current_image;
  public void init ()
    {
    try
      {
      num_images = Integer.parseInt (getParameter ("NUM_IMAGES"));
      }
    catch (NumberFormatException nfe)
      {
      num_images = 0;
      }
    tracker = new MediaTracker (this);
    images = new Image [num_images];
    for (int i = 0; i < num_images; i ++)
      {
      images [i] = getImage (getDocumentBase (), "Image" + i + ".gif");
      tracker.addImage (images [i], i);
      }
    }
```

AWT Overview; Multimedia

```
...
  public void run ()
    {
    for (int i = 0; i < num_images; i ++)
      {
      showStatus ("Loading image: " + i + ".");
      try { tracker.waitForID (i); } catch (InterruptedException ie) { }
      if (tracker.isErrorID (i))
        {
        showStatus ("Error loading image: " + i + ". Quitting.");
        return;
        }
      }
    this.showStatus ("Loading images: done.");
    for ( ; ; )
      {
      if (++ current_image >= images.length) current_image = 0;
      getGraphics ().drawImage (images [current_image], 0, 0, this);
      getToolkit ().sync ();  // draw ***now***
      try ( Thread.sleep (200); } catch (InterruptedException ie) { }
      }
    }
```

AWT Overview; Multimedia

- Double buffering
  - draw to memory instead then ‘blast’ to the actual screen
  - gives smoother, apparently faster drawing

```
// tile image across an off-screen buffer
Image buffered_image = createImage (client_width, client_height);
Graphics bg = buffered_image.getGraphics ();
doLotsOfIntricateDrawingInto (bg);
bg.dispose ();
…
// now draw onto the screen proper
g.drawImage (buffered_image, 0, 0, null);
```

- Clipping
  - record amount of screen that needs to be redrawn and only actually redraw that area
    - g.clipRect (x, y, w, h)
  - less work, so faster

AWT Overview; Multimedia

- Toolkit
  - *"'glue' code joining the platform-independent classes in the java.awt package with their counterparts in java.awt.peer"*
  - Component.getToolkit ()
  - [mostly] not for mere mortals to use
    - some methods do have training wheels, however:
      - sync
        - ensures that the display is up-to-date (for animation)
      - beep
      - getScreenSize
      - getSystemClipboard
      - getPrintJob
  - Applets can't use (some parts of) the toolkit

AWT Overview; Multimedia

- Java Media Framework
  - media capture, playback and conferencing APIs
  - being developed by Sun, Silicon Graphics & Intel
  - not yet mature, but maturing fast
  - audio: μLAW, AIFF, WAV
  - movie: MPEG-{1, 2}, AVI, QuickTime
  - MIDI: SG only, others RSN
  - interfaces to native OS facilities for efficiency
  - same kind of delayed activity as for images
    - same reason!