# Java Graphics and Applets

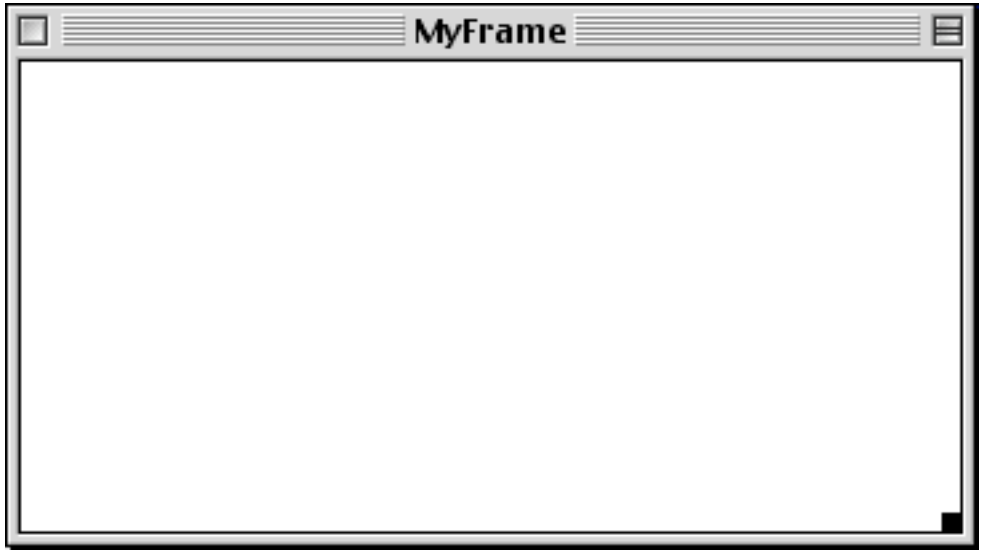
# Java Graphics

- Two main Graphics libraries:
  - java.awt (Abstract Window Toolkit)
    - older
    - accepted by most web browsers
  - java.swing
    - newer
    - more portable (less sensitive to window system)
    - might not be support on your browser, without plugins

# We will use awt

- relatively simple to use
- works on most browsers

# class *Frame: Basic "window"*



# Drawing on Objects

- In order to draw in a frame or other objects, an Object of class

  Graphics

  must first be obtained for the frame. This is referred to as a graphics **context**. This is done using the call

  **`getGraphics()`**

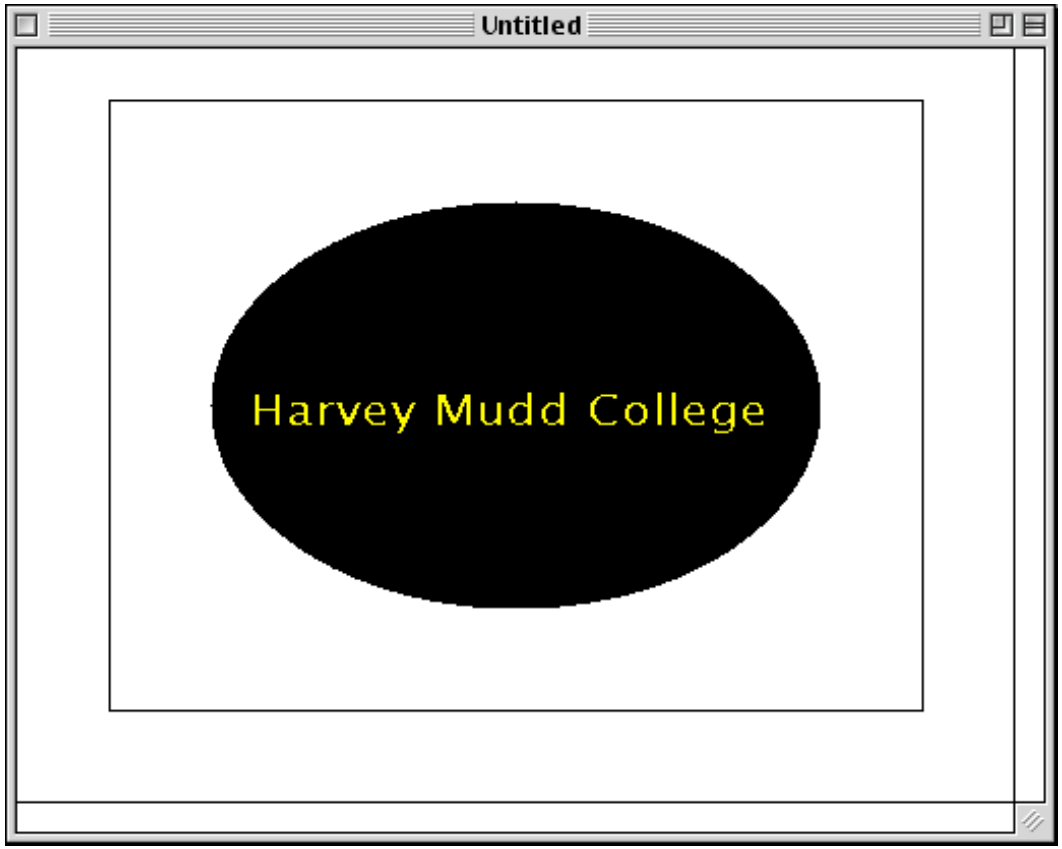
  which returns a Graphics object.

# *abstract* class *Graphics*

- Think of a Graphics object as being like a **canvas** on which drawing takes place.
- The actual display of what is drawn is usually done behind the scenes, without giving an explicit command to display the Graphics object.
- Graphics objects have other uses too, e.g. printing.

# A Simple Graphics Program

# Desired Result



```
import java.awt.*;                          // to get awt

public class MyFrame1 extends Frame     // Customize Frame class
{

// test Program

public static void main(String[] arg)
  {
  new MyFrame1("My Frame", 50,  50);
  }


// construct frame with title and position

MyFrame1(String title, int x, int y)
  {
  setTitle(title);

  setBackground(Color.white);             // set the background color

  reshape(x, y, 500, 400);                // set the location and size

  setVisible(true);                       // show the frame

  toFront();
  }
```

```
// draw items in Graphics in Frame; called by paint()

void drawStuff()
  {
  Graphics g = getGraphics();
  g.setColor(Color.black);                  // set the color for drawing
  g.drawRect(50, 50, 400, 300);             // draw a rectangle
  g.fillOval(100, 100, 300, 200);           // fill an oval

  g.setFont(new Font("Times", Font.BOLD, 20));
  g.setColor(Color.yellow);
  g.drawString("Harvey Mudd College", 120, 210);  // draw a string
  }


// paint(Graphics) will be called by the system to paint the Frame when
// necessary, e.g. when setVisible(true) is called.
// This call is done implicitly; we do not see it in the source.
// The Graphics of the frame will then be passed as an argument.

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

# Typical awt Drawing Methods

- `void drawLine(int x1, int y1, int x2, int y2)`
- `void drawRect(int x, int y, int width, int height)`
- `void drawOval(int x, int y, int width, int height)`
- `void drawPolygon(int[] xPoints, int[] yPoints, int nPoints)`
- `void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)`
- `void drawString(String str, int x, int y)`
- `void drawImage(Image img, int x, int y, Color bgcolor, ImageObserver observer)`
- Blue bullets also have fill instead of draw.

# Frame Buffering

This may be somewhat confusing.

Follow the lead of canned examples when in doubt, which may be most of the time.

This is a legacy from the Java designers.

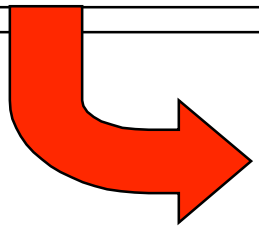
# paint() method

some **event** occurs, such as setting Frame visible, or `repaint()` called explicitly

`update()` is called on Graphics of Frame, resulting in:

Frame being **cleared** with **background** color, then

`paint()` is called on Graphics of Frame



whatever is drawn in paint is displayed

The `update()` method can be over-ridden.

# Flicker Prevention 1

- Clearing the background in `update()` can create lots of **flicker** in the application.

- It is customary to **over-ride** `update()` in the customized frame as follows:

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

# Flicker Prevention 2

- Instead of painting the background, then drawing on it, it is better to "paint" a complete image covering the visible area of the Frame.
- This image is known as an

  **off-screen buffer**

  because the buffer is not part of the Frame.
- The buffer is drawn on prior to `update()`.
- The buffer contents is then painted in `update()`.

```
public class MyFrame2 extends Frame    // Customize Frame class
{
Image buffer;
```

No Flicker 2: off-screen buffer

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

No Flicker 1: over-ride update()

```
void drawStuff()
  {
  Graphics g = buffer.getGraphics();
  ... as before ...
  }
```

draw on buffer Graphics, not on Frame Graphics

```
public void paint(Graphics g)
  {
  if( buffer == null )
    buffer = createImage(getWidth(), getHeight());
```

create buffer on demand

```
  drawStuff();
```

could also be done elsewhere

```
  g.drawImage(buffer, 0, 0, null);
  }
```

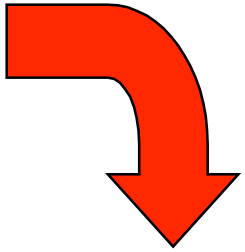
paint the buffer into the frame

# Flicker Prevention Summary

some event occurs, such as setting Frame visible, or `repaint()`

`update()` is called on Graphics of Frame, resulting in:

`paint()` is called on Graphics of Frame



**drawing** is on **buffer**

**buffer** is painted on Graphics of Frame

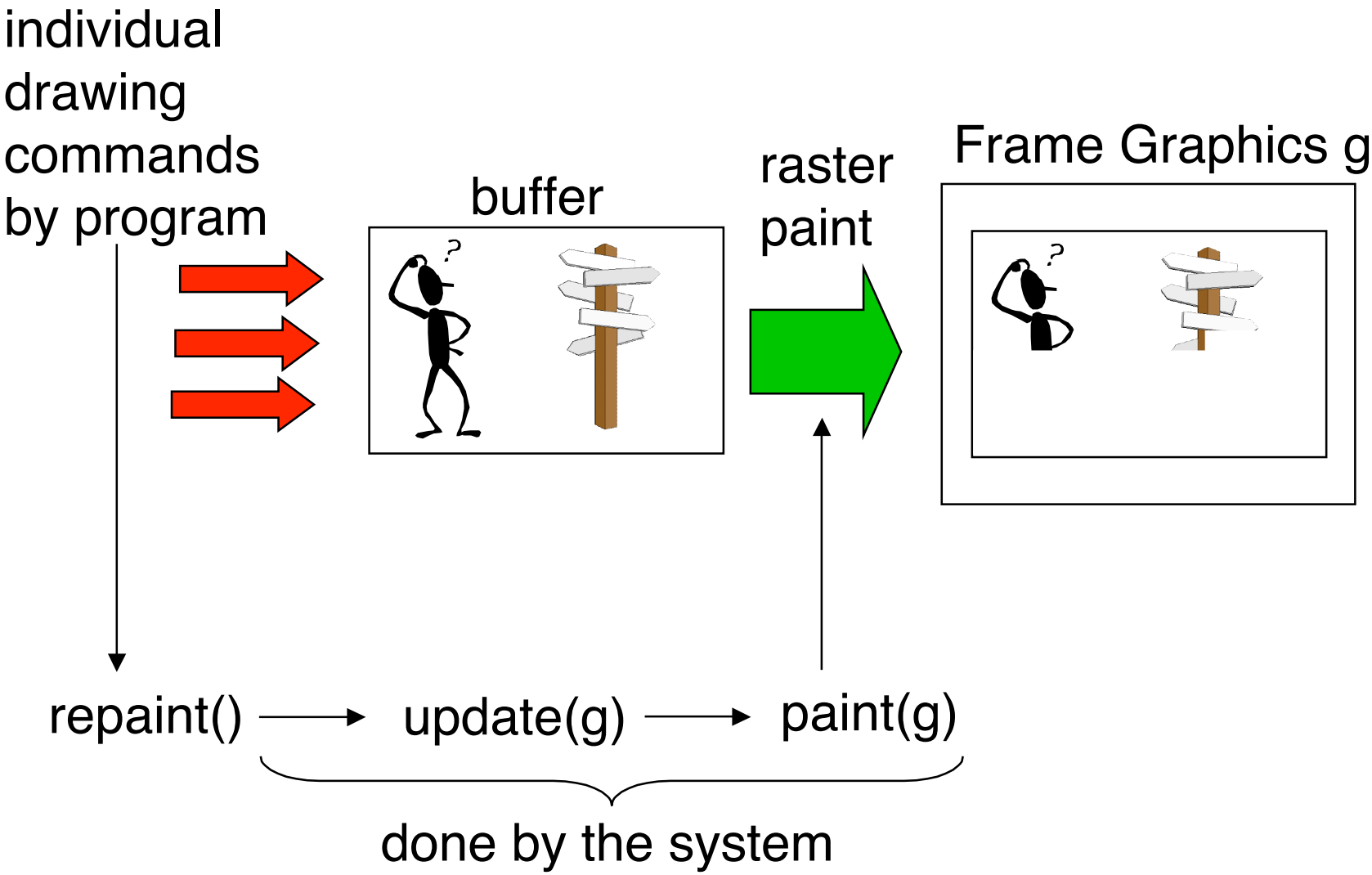
# repaint()

- When the programmer wants to force repainting of the Frame, she calls

  **repaint()**

- This causes the system to schedule a call to **update()**.
- **repaint()** has no arguments.
- The programmer normally does not call **paint()** directly, outside of **update()**.

# repaint() -> paint()



# Applets

- "Applet" means "small application"
- Run in one of two ways:
  - In a web browser
  - Using a program appletviewer
- Applet runs in a specialized Frame (type of window)
- No `main()`; instead: `init()`, `run()`.

# Viewing Applets on the Web or using appletviewer

**contents of MyApplet1.html:**

```
<html>
<title>MyApplet1</title>
<head>
<!-- MyApplet1 applet-->
</head>
<body>

<applet code=MyApplet1.class width=500 height=400></applet>

</body>
</html>
```


**run by:**

```
appletviewer MyApplet1.html
```

# Web Applet Restrictions

- Might not be able to load *files* on server or client
- Instead load content of URL's, but maybe only from the same server that contains the applet code

# Examples

- appletGraphics example on our web pages
- and others

# CS60 BaseApplet

- The class BaseApplet was created to provide functionality of a type useful in CS 60 assignments.
- It sets up the off-screen buffering.
- It provides a real-time loop:
  - Program over-rides step() to have her method called in the loop.
  - Programmer over-rides continue() to possibly terminate the loop.
- Programmer over-rides mouse methods to get mouse input.
- Examples of using menu inputs are provided.