

---

# Graphics Applets

By

Mr. Dave Clausen

# Applets

- A Java application is a stand-alone program with a `main` method
- A Java *applet* is a program that is intended to be transported over the Web and executed using a web browser
- An applet also can be executed using the Applet Viewer tool of the Java Software Development Kit
- An applet doesn't have a `main` method
- Instead an applet uses `public void init()` method
- Applets can contain:
  - Methods you define
  - Variables and constants
  - Decisions, loops, and arrays

# Create an Applet

- Use JCreator to create the java and html files
  - Write applet source code in Java
    - ◆ Save with `.java` file extension
  - Compile applet into bytecode
  - Write HTML document
    - ◆ Save with `.html` or `.htm` file extension
    - ◆ Include a statement to call the **compiled** Java class (`.class`)
- To “run” the applet in JCreator
  - **Compile** the **java** code
  - **Execute** the **html** code to view the applet in the Applet Viewer
- Or load HTML document into a Web browser
  - When you make changes, save the java code, recompile the java code, and refresh the browser. (LCUSD has this blocked at school.)

# Inheritance and bytecode

- The class that defines an applet *extends* the `Applet` class
- This makes use of *inheritance*.
- An applet is embedded into an HTML file using a tag that references the bytecode (`.class`) file of the applet class
- The bytecode version of the program is transported across the web and executed by a Java interpreter that is part of the browser

# HTML Comments

➤ Comments begin with `<!--` (no spaces between)

➤ Comments end with `-->`

```
<!--*****  
*  
* Mr. Clausen      9999  
*  
* Program Move Circle Applet Animation  
*  
* AP Computer Science Java Period ?  
*  
* Starting Date: 5/?/200?      Due Date: 5/?/200?  
*  
* This program will animate a circle in a Java Applet  
* Don't forget to include comments describing your applet and  
* what exactly it does.  
*****-->
```

# HTML Template & applet Tag

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
```

```
  <head>
```

```
    <title>YourLastName FirstName ID# Final Project</title>
```

```
  </head>
```

```
  <body>
```

```
    <center> <h3>YourLastName FirstName ID# Final Project</h3>
```

```
    <applet code="LastNameFirstNameFP.class"  
           width=760 height=520>
```

```
  </applet>
```

```
  </center>
```

```
</body>
```

```
</html>
```

An HTML Template For Graphics Programs

# Applet Class Methods

- Our Java Source code public class needs to include **extends** Applet (for example)
  - `public class MoveCircle extends Applet`
- Applet class methods Inherited from the Applet Class
  - Automatically Invoked by the Web browser when the browser runs the applet
  - These methods don't have to be included in your applet unless you wish to override the methods in the parent class.
  - `public void init() //use this method`
  - `public void start()`
  - `public void stop()`
  - `public void destroy()`

# Applet method Execution

## ➤ `init()` method

- Executes when a Web page containing an `Applet` is loaded
- Or when running `appletviewer` command

## ➤ `start()` method

- Executes after `init()` method
- Executes again every time the applet becomes active after it has been inactive

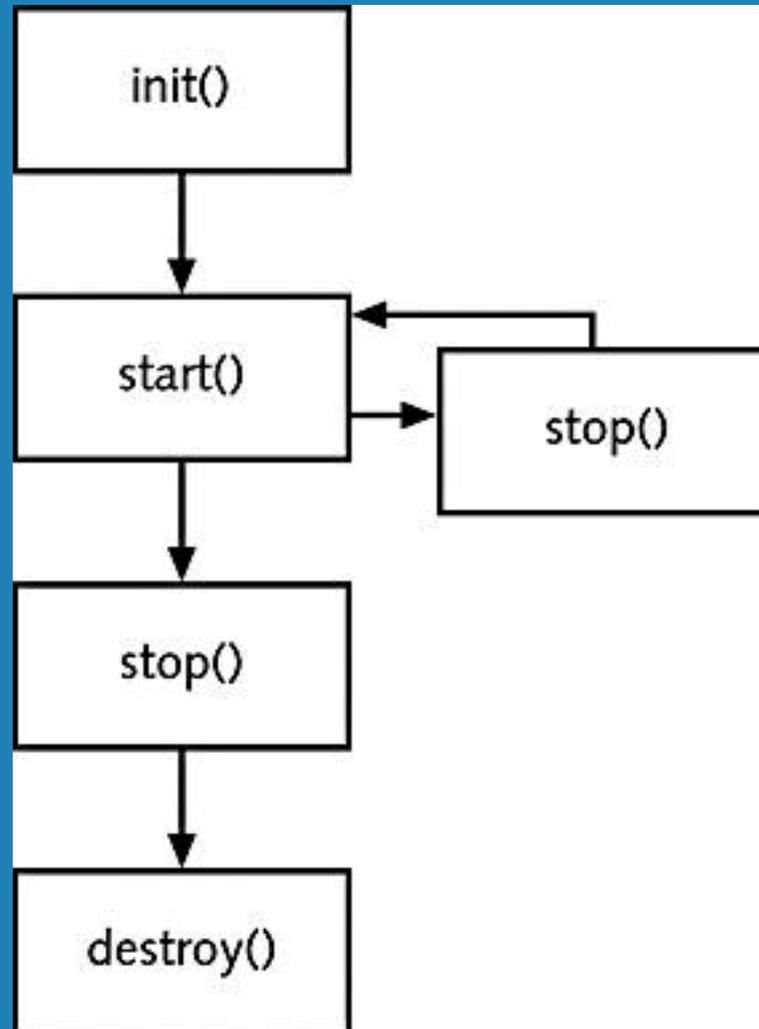
## ➤ `stop()` method

- Invoked when user leaves Web page

## ➤ `destroy()` method

- Called when user closes browser or Applet Viewer
- Releases any resources `Applet` might have allocated

# Applet Life Cycle



# Overriding applet Methods

- Overriding a method means
  - Replace original version (the inherited version)
- Advanced programmers may choose to override the `stop ()` and `destroy ()` methods
  - We will not override them
- We will override the `init ()` method
  - ⚡ resize our applet
  - ⚡ set the background color
  - For example:

```
public void init()  
{  
    //Set the size of the applet panel  
    resize(760, 520);  
    setBackground (Colorwhite);  
}
```

# Additional Applet Methods

- There are nearly 200 additional methods
  - Manipulate components within Japplets or Applets
  - We are not using applet components in our programs
  - Components include
    - ◆ Buttons
    - ◆ Labels and
    - ◆ Text Fields
  - Read definitions at *<http://java.sun.com>* Web site

# Applet paint Method

- There are several other special methods that serve specific purposes in an applet.
- The `paint` method, for instance, is executed automatically and is used to draw the applet's contents
- The `paint` method accepts a parameter that is an object of the `Graphics` class
  - `public void paint(Graphics g)`
- A `Graphics` object defines a *graphics context* on which we can draw shapes and text
- The `Graphics` class has several methods for drawing shapes

# paint () Method

## ➤ paint () method

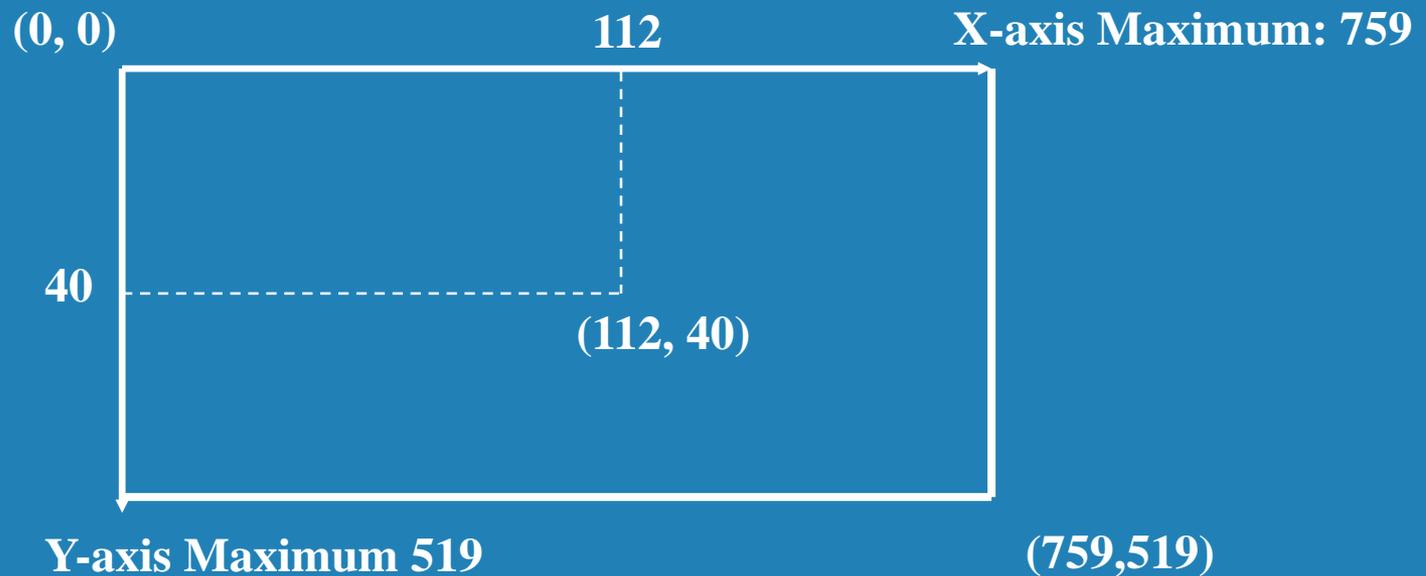
- Runs when Java displays the applet
- We will write our own method to override the default method
- Executes automatically every time someone
  - ◆ Minimizes, maximizes, or resizes Applet Viewer window or browser window
- Method header
  - ◆ `public void paint(Graphics g)`

# Drawing Shapes

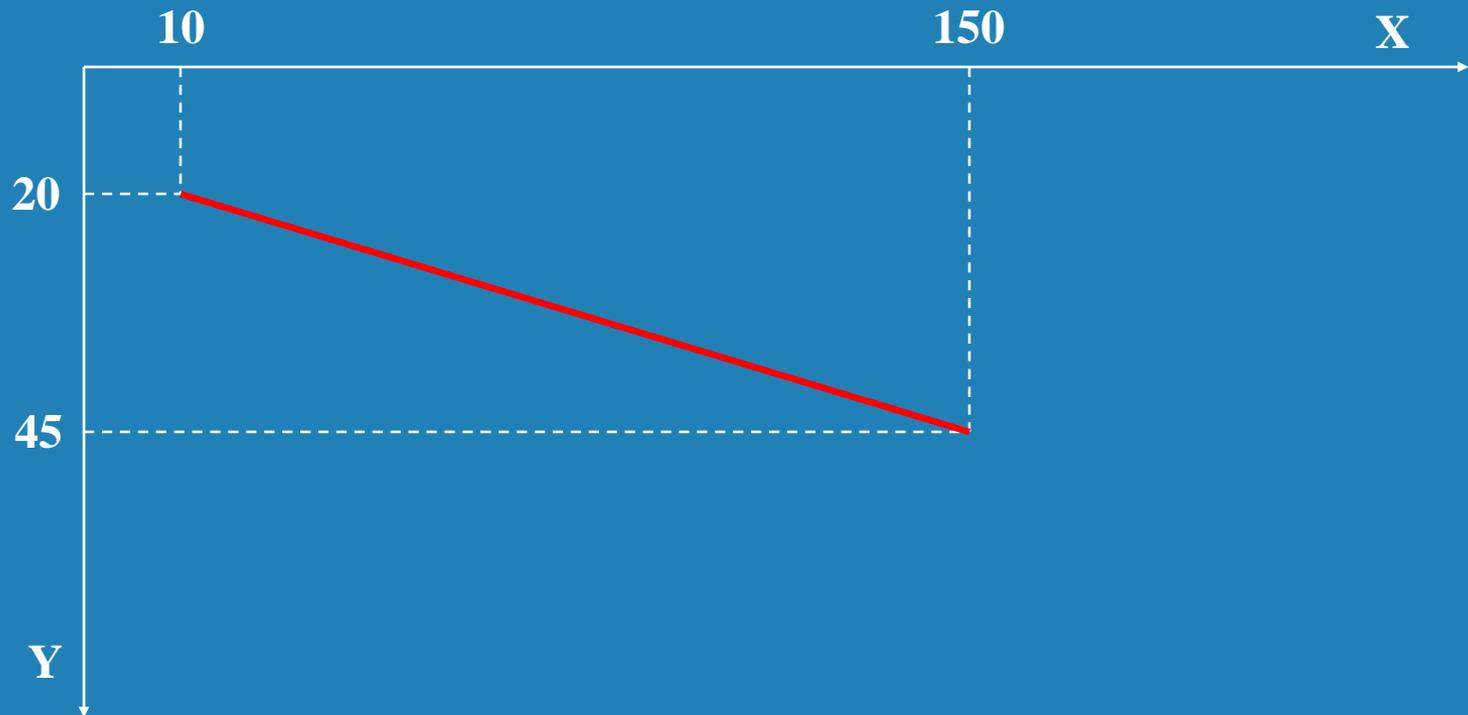
- Let's explore some of the methods of the `Graphics` class that draw shapes in more detail
- A shape can be filled or unfilled, depending on which method is invoked
- The method parameters specify coordinates and sizes
- Recall that the Java coordinate system has the origin in the top left corner
- Shapes with curves, like an oval, are usually drawn by specifying the shape's *bounding rectangle*
- An arc can be thought of as a section of an oval

# Coordinate Systems

- Each pixel can be identified using a two-dimensional coordinate system
- When referring to a pixel in a Java program, we use a coordinate system with the origin in the top-left corner



# Drawing a Line

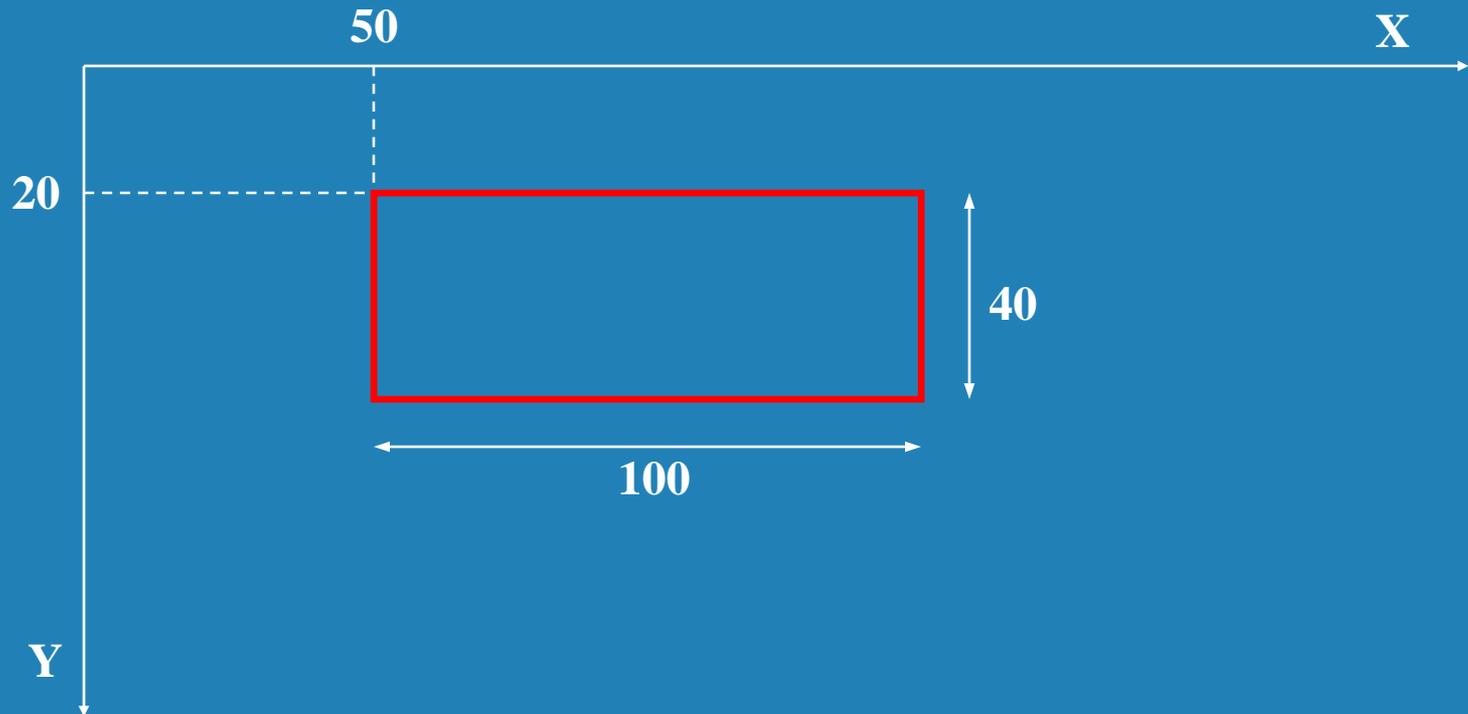


```
g.drawLine (10, 20, 150, 45);
```

or

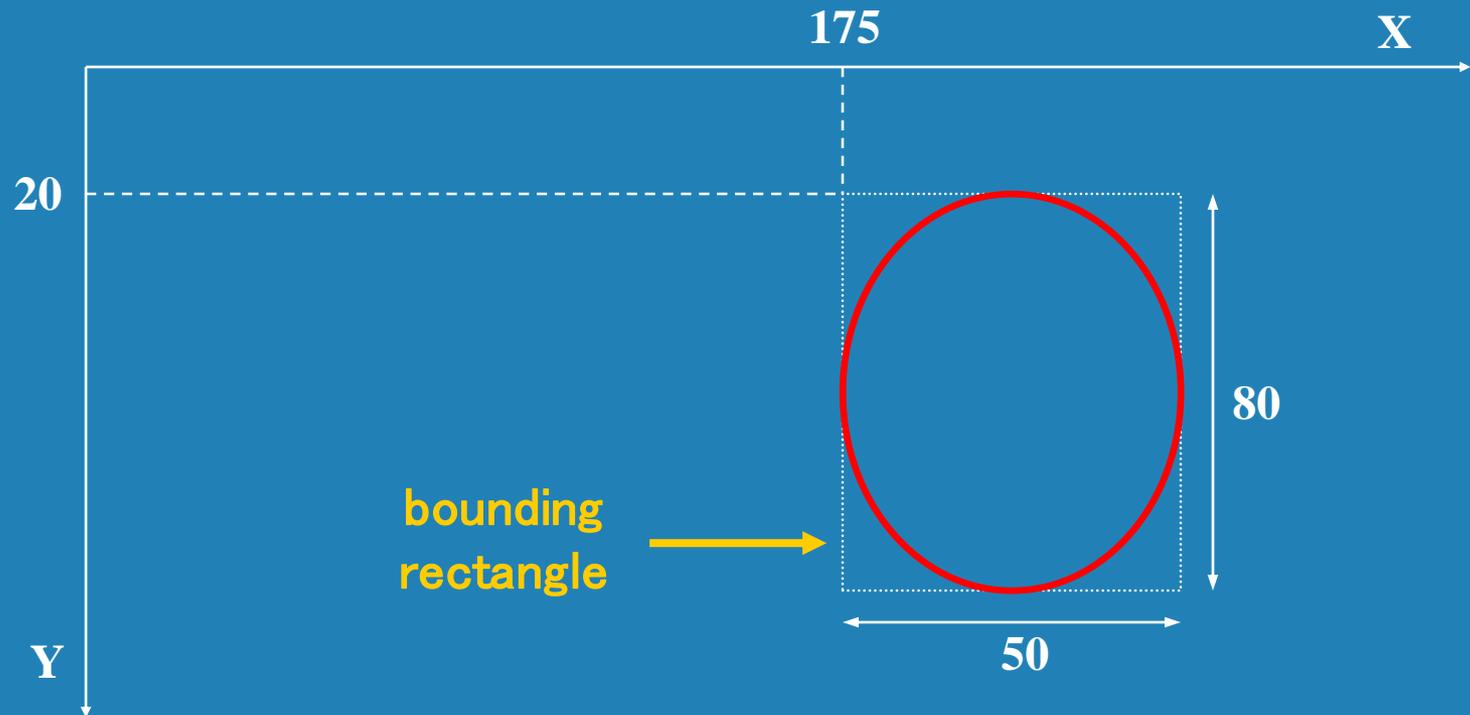
```
g.drawLine (150, 45, 10, 20);
```

# Drawing a Rectangle



```
g.drawRect (50, 20, 100, 40);
```

# Drawing an Oval



```
g.drawOval (175, 20, 50, 80);
```

# Drawing a Polygon

- [drawPolygon](#)(int[ ] xPoints, int[ ] yPoints, int nPoints)
  - Draws a closed polygon defined by arrays of  $x$  and  $y$  coordinates.
- [fillPolygon](#)(int[ ] xPoints, int[ ] yPoints, int nPoints)
  - Fills a closed polygon defined by arrays of  $x$  and  $y$  coordinates.
- fillPolygon example
  - [HTMLfile](#)
  - [Java file](#)

```
public class DrawPolygon extends Applet {  
    ...  
    public void paint (Graphics g) {  
        int[ ] xPoints = {10, 80, 10, 10};  
        int[ ] yPoints = {120, 160, 200, 120};  
        g.setColor (Color.orange);  
        g.fillPolygon (xPoints,yPoints,4);  
    }  
}
```

# Shape Methods Summary

**clearRect**(int x, int y, int width, int height)

Clears the specified rectangle by filling it with the background color of the current drawing surface.

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

Draws a 3-D highlighted outline of the specified rectangle.

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

Draws the outline of a circular or elliptical arc covering the specified rectangle.

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

Draws a line, using the current color, between the points (x1, y1) and (x2, y2) in this graphics context's coordinate system.

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

Draws the outline of an oval.

**drawPolygon**(int[] xPoints, int[] yPoints, int nPoints)

Draws a closed polygon defined by arrays of x and y coordinates.

**drawPolygon**(Polygon p)

Draws the outline of a polygon defined by the specified Polygon object.

**drawRect**(int x, int y, int width, int height)

Draws the outline of the specified rectangle.

**drawRoundRect**(int x, int y, int width, int height, int arcWidth, int arcHeight)

Draws an outlined round-cornered rectangle using this graphics context's current color.

**drawString**(String str, int x, int y)

Draws the text given by the specified string, using this graphics context's current font and color.

# Shape Methods Summary 2

**drawString**(String str, int x, int y)

Draws the text given by the specified string, using this graphics context's current font and color.

**fill3DRect**(int x, int y, int width, int height, boolean raised)

Paints a 3-D highlighted rectangle filled with the current color.

**fillArc**(int x, int y, int width, int height, int startAngle, int arcAngle)

Fills a circular or elliptical arc covering the specified rectangle.

**fillOval**(int x, int y, int width, int height)

Fills an oval bounded by the specified rectangle with the current color.

**fillPolygon**(int[] xPoints, int[] yPoints, int nPoints)

Fills a closed polygon defined by arrays of *x* and *y* coordinates.

**fillPolygon**(Polygon p)

Fills the polygon defined by the specified Polygon object with the graphics context's current color.

**fillRect**(int x, int y, int width, int height)

Fills the specified rectangle.

**fillRoundRect**(int x, int y, int width, int height, int arcWidth, int arcHeight)

Fills the specified rounded corner rectangle with the current color.

# Drawing Arcs

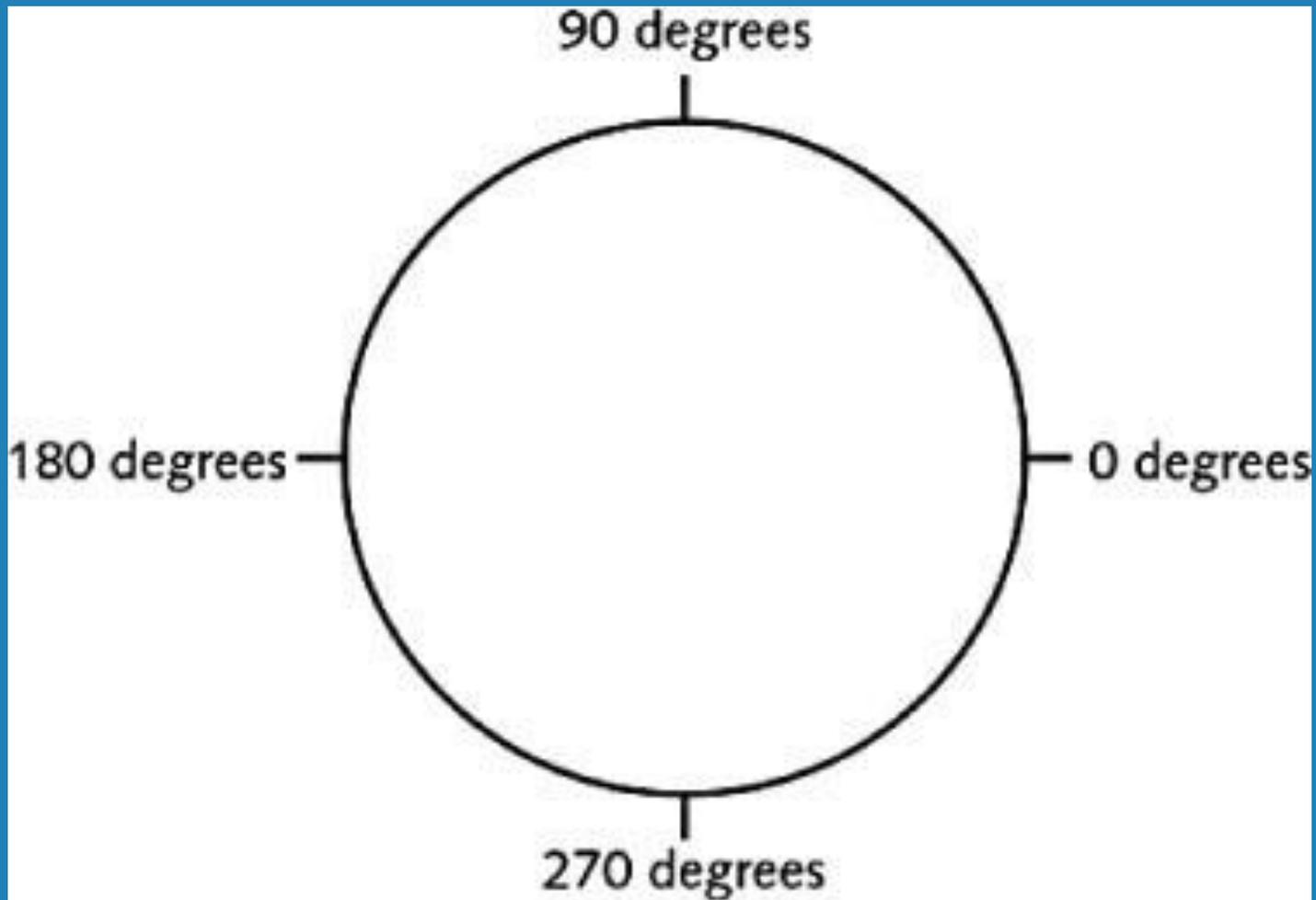
## ➤ `drawArc()` method arguments

- x-coordinate of upper-left corner of imaginary rectangle that represents bounds of imaginary circle that contains arc
- y-coordinate of same point
- Width of imaginary rectangle that represents bounds of imaginary circle that contains arc
- Height of same imaginary rectangle
- Beginning arc position
- Arc angle

## ➤ `drawArc`(int x, int y, int width, int height, int startAngle, int arcAngle)

- Draws the outline of a circular or elliptical arc covering the specified rectangle.

# Arc Angles



# fillArc method

## ➤ fillArc () method

- Creates a solid arc

- ♦ Two straight lines are drawn from the arc endpoints to the center of an imaginary circle whose perimeter the arc occupies.

## ➤ fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)

- Fills a circular or elliptical arc covering the specified rectangle

# Polygons

## ➤ `drawPolygon()` method

- Draws complex shapes
- Requires three arguments
  - ◆ An integer array that holds the x-coordinate positions
  - ◆ A second array that holds the corresponding y-coordinate positions
  - ◆ The number of pairs of points to connect
  - ◆ `drawPolygon`(int[] xPoints, int[] yPoints, int nPoints)
    - Draws a closed polygon defined by arrays of x and y coordinates.

# fillPolygon Method

## ➤ fillPolygon() method

- Draws a solid shape
- The beginning and ending points need to be the same to “close” the shape.
  - ◆ Therefore, there will be one more set of ordered pairs than the number of sides you wish to draw
- fillPolygon(int[] xPoints, int[] yPoints, int nPoints)
- Fills a closed polygon defined by arrays of  $x$  and  $y$  coordinates.

# clearRect method

- clearRect(int x, int y, int width, int height)
- Clears the specified rectangle by filling it with the background color of the current drawing surface.
  - Appears empty or “clear”
- We can use clearRect to erase individual items or the entire background scene if your animation has more than one background scene.

# Color

- A Java programmer can control the color of images by using the Color class.
- The Color class is included in the package `java.awt`.
- The Color class provides the class constants shown in Table 19-2.
- The Graphics class includes two methods for examining and modifying an image's color (Table 19-3).

# Color Methods

The Graphics class includes two methods for examining and modifying an image's color

(Table 19-3)

METHOD	WHAT IT DOES
<code>Color getColor()</code>	Returns the current color of the graphics context.
<code>void setColor(Color c)</code>	Sets the color of the graphics context to c.

## ·Using a predefined color

·`g.setColor (Colorred); // red is a method of the color class`

# Color Constants

<b>COLOR CONSTANT</b>	<b>COLOR</b>
<code>public static final Color red</code>	red
<code>public static final Color yellow</code>	yellow
<code>public static final Color blue</code>	blue
<code>public static final Color orange</code>	orange
<code>public static final Color pink</code>	pink
<code>public static final Color cyan</code>	cyan
<code>public static final Color magenta</code>	magenta
<code>public static final Color black</code>	black
<code>public static final Color white</code>	white
<code>public static final Color gray</code>	gray
<code>public static final Color lightGray</code>	light gray
<code>public static final Color darkGray</code>	dark gray

Table 19-2: Color Class Constants

# Create Your Own Colors

- Every color can be represented as a mixture of the three additive primary colors **Red**, **Green**, and **Blue**
- In Java, each color is represented by three numbers between 0 and 255 that collectively are called an *RGB value*
- A color is defined in a Java program using an object created from the `Color` class

# The Color Class

- The `Color` class contains several static predefined colors. Here are a few of the color constants with their RGB values.

<u>Object</u>	<u>RGB Value</u>
<code>Color.black</code>	<code>0, 0, 0</code>
<code>Color.blue</code>	<code>0, 0, 255</code>
<code>Color.cyan</code>	<code>0, 255, 255</code>
<code>Color.orange</code>	<code>255, 200, 0</code>
<code>Color.white</code>	<code>255, 255, 255</code>
<code>Color.yellow</code>	<code>255, 255, 0</code>

# How To Create Your Own Colors

- Java allows the programmer more refined control over colors by using RGB (red/green/blue) values.
- In this scheme, there are:
  - 256 shades of red
  - 256 shades of green
  - 256 shades of blue
- The programmer "mixes" a new color by selecting an integer from 0 to 255 for each color and passing these integers to a Color constructor as follows:

```
new Color (<int for red>, <int for green>, <int for blue>)
```

# Custom Color Examples

## ➤ Examples of creating and **instantiating** custom colors

- `Color myGreen = new Color (0, 204, 0);`
- `Color myPurple = new Color (153, 0, 150);`
- `Color myBrown = new Color (166, 124, 82);`
- `Color myOrange = new Color (251, 136, 93);`

## ➤ **Using a predefined color**

- `g.setColor (Colorred);`  
`// red is a method of the color class`

## ➤ **Using your custom color**

- `g.setColor (myGreen);`

# Create Random Colors

The next code segment shows how to create a random color with RGB values:

```
// Create a random color from randomly
    generated RGB values

int r = (int) (Math.random() * 256);

int g = (int) (Math.random() * 256);

int b = (int) (Math.random() * 256);

Color randomColor = new Color (r, g, b);
```

# The Color Class

- Every drawing surface has a *background color*

*setBackground (Colorwhite);*

- Every graphics context has a current *foreground color*

*g.setColor (Colorblue);*

- Both can be set

- See [Snowman.java](#) and [Snowman.html](#)

- Before starting animation, experiment with drawing shapes in a “still life” using Snowman.java as an example in a “paint” method.

# Text Properties

- A text image has several properties, as shown in Table 19–8 below
- These are set by adjusting the color and font properties of the graphics context in which the text is drawn.

TEXT PROPERTY	EXAMPLE
Color	Red, green, blue, white, black, etc.
Font style	Plain, <b>bold</b> , <i>italic</i>
Font size	10 point, 12 point, etc.
Font name	Courier, Times New Roman, etc.

# Text Properties

## The Font Class

- An object of class Font has three basic properties:
  - a name
  - a style
  - and a size
- The following code creates one Font object with the properties Courier bold 12 and another with the properties Arial bold italic 10:
- Use descriptive names for your fonts when you **instantiate** them as illustrated below

```
Font courierBold12      = new Font("Courier", Font.BOLD, 12);  
Font arialBoldItalic10 = new Font("Arial", Font.BOLD + Font.ITALIC, 10);
```

# Text Properties

- The Font constants `PLAIN`, `BOLD`, and `ITALIC` define the font styles.
- The font size is an integer representing the number of points, where one point equals  $1/72$  of an inch.
- The available font names depend on your particular computer platform.

# Text Properties

- Table 19–9 lists the principal font methods:

FONT METHOD	WHAT IT DOES
<code>public Font(String name,           int style,           int size)</code>	Creates a new Font object with the specified properties; style must be PLAIN, BOLD, ITALIC, or a combination of these using +.
<code>public String getName()</code>	Returns the current font name.
<code>public int getStyle()</code>	Returns the current font style.
<code>public int getSize()</code>	Returns the current font size.

# Text Properties

## ➤ Setting the Color and Font Properties of Text

➤ Assume that we want to display the text "Hello world!" in red with the font Courier bold 14. The following code would do this:

```
Font courierBold14= new Font ("Courier", Font.BOLD, 14);  
  
g.setColor (Color.red);  
g.setFont (courierBold14);  
g.drawString ("Hello world!", 100, 100);
```

➤ Changing the font and color of a graphics context affects all subsequent graphics operations in that context but does not alter the font or color of existing images.

# Applet Methods Review

- In previous examples we've used the `paint` method of the `Applet` class to draw on an applet
- The `Applet` class has several methods that are invoked automatically at certain points in an applet's life
- The `init` method, for instance, is executed only once when the applet is initially loaded
- The `start` and `stop` methods are called when the applet becomes active or inactive
- The `Applet` class also contains other methods that generally assist in applet processing

# repaint () Method

## ➤ repaint () method

- We don't call the `paint ()` method directly
- We call the `repaint ()` method when the window needs to be updated, perhaps with new images.
- The `repaint ()` method calls another method named `update ()` which in turn calls the `paint ()` method.
- Creates `Graphics` object

# Animations

- An animation is a series of images that gives the appearance of movement (24 frames per second)
- To create the illusion of movement, we use a delay to change the scene after an appropriate amount of time or to slow down the speed of the moving object.
- Start by declaring a constant:
- `private final int SLEEP_TIME = 10;`

# Animations continued

- Include this code for the delay:

```
//delay  
  
    try  
    {  
        Thread.sleep(SLEEP_TIME);  
    }  
  
    catch(InterruptedException e)  
    {  
    }  
}
```

# Sources

## ➤ Java Software Solutions

- by Lewis and Loftus
- Addison-Wesley

## ➤ Fundamentals of Java Second Edition

- by Lambert and Osborne
- South-Western

## ➤ Java Programming (versions 1, 2, & 4)

- by Joyce Farrell
- Thomson