Java

Graphics

Thinking is the hardest work there is. Which is probably the reason why so few engage in it. Henry Ford

Lecture objectives

- To be able to understand the following fundamental concepts of the Java programming language:
  - introduction to applets
  - graphical methods

Introduction to Graphics

- Many computer programs have graphical components
- A picture is broken down into pixels (picture elements)
  - these individual dots that make up the image on your video monitor,
  - each pixel is stored separately

Creating a Drawing

- Java's Abstract Window Toolkit provides classes that are used to build programs that use graphics

```
import java.awt.*;
```

- The `Graphics` class, of the `java.awt` package, makes it possible to create line drawings on the screen.

- This class provides methods for
  - drawing lines, rectangles, ovals, arcs, polygons, and polylines.
  - displaying text in various ways

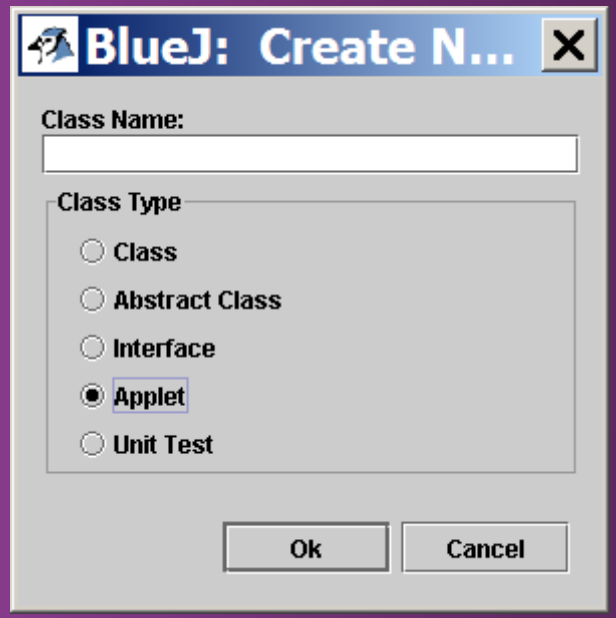
Applications and Applets

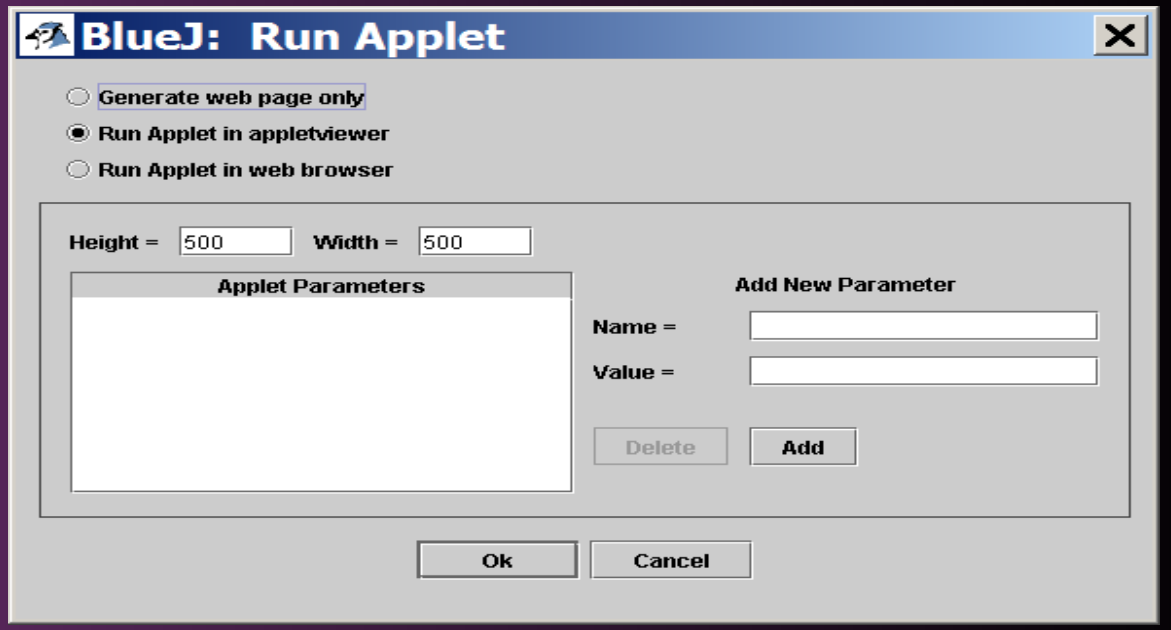
- There are two kinds of Java programs:
  - Java application: a stand-alone program with a `main` method
  - Java applet: a Java program that is intended to transported over the web and executed using a web browser

- An applet can also be executed using the appletviewer tool of BlueJ

- Applets provide a simple way to introduce graphics

Applets and BlueJ





Applets

- We import the Applet class from the java.applet package

```
import java.applet.Applet;
```

- An applet doesn't have a `main` method
  - because a web browser that executes it is already running
  - Applets can be thought of as part of a larger program
- It has several methods that serve specific purposes
- The `paint` method is automatically executed and is used to draw the applets contents

Applets and Inheritance

- The class that defines the applet *extends* the Applet class

```
public class Einstein extends Applet
```

- An applet is an example of inheritance

Graphics Contexts

- An instance of the `Graphics` class is a rectangular area
  - This is called a ***graphics context***
- The `paint` method can alter pixels in the *graphics context*
  - it accepts a parameter that is an object of the `Graphics` class

```
public void paint (Graphics page)
public void paint (Graphics g)
```

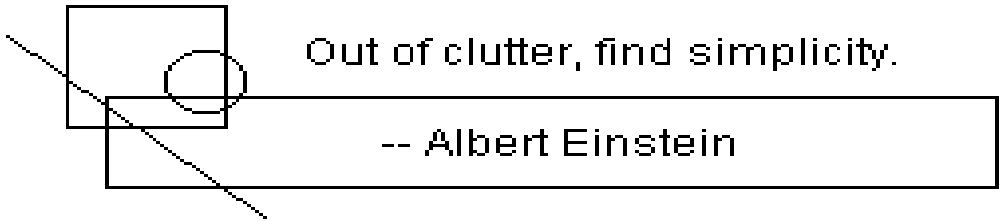
- The `Graphics` class provides many methods for drawing graphics

```
//*******************************************************************
//  Einstein.java       Author: Lewis and Loftus
//  Demonstrates a  simple applet
//*******************************************************************
import java.applet.Applet;
import java.awt.*;

public class Einstein extends Applet
{
   //-----------------------------------------------------------
   //  Draws a quotation by Albert Einstein among some shapes.
   //-----------------------------------------------------------
   public void paint (Graphics page)
   {
       page.drawRect (50, 50, 40, 40);    // square
       page.drawRect (60, 80, 225, 30);   // rectangle
       page.drawOval (75, 65, 20, 20);    // circle
       page.drawLine (35, 60, 100, 120);  // line
       page.drawString ("Out of clutter, find simplicity.", 110, 70);
       page.drawString ("-- Albert Einstein", 130, 100);
    }
}
```

Applet Viewer: Einstein.class

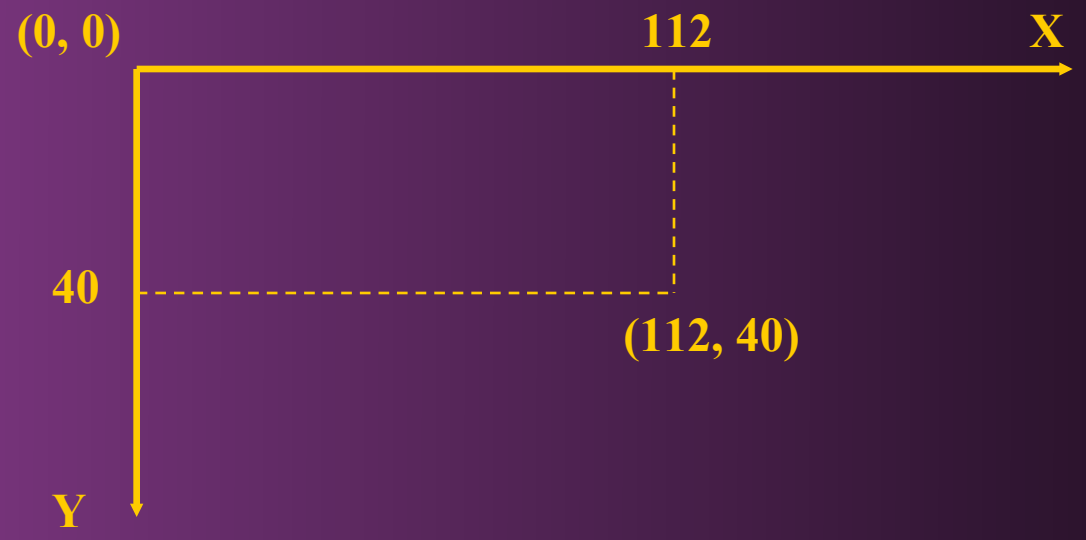
Applet



Applet started.

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 upper left corner



Using Graphics Methods

- If `page` is a graphics context, the pixels in `page` can be changed by calling one of the drawing methods in the `Graphics` class.

  public void paint (Graphics page)

- The `drawLine` method draws a line from one point to another:

  page.drawLine(x1, y1, x2, y2);

  (`x1`, `y1`) is one of the line's endpoints; (`x2`, `y2`) is the other.

Drawing a Line

page.drawLine (10, 20, 150, 45);

or page.drawLine (150, 45, 10, 20);

Rectangles

- There's one method to draw an outline and another to draw a filled rectangle.
- Rectangle methods take four arguments (or more for rounded rectangles):
  - the first two specify the rectangle's upper left corner.
  - the next two specify the rectangle's width and height.

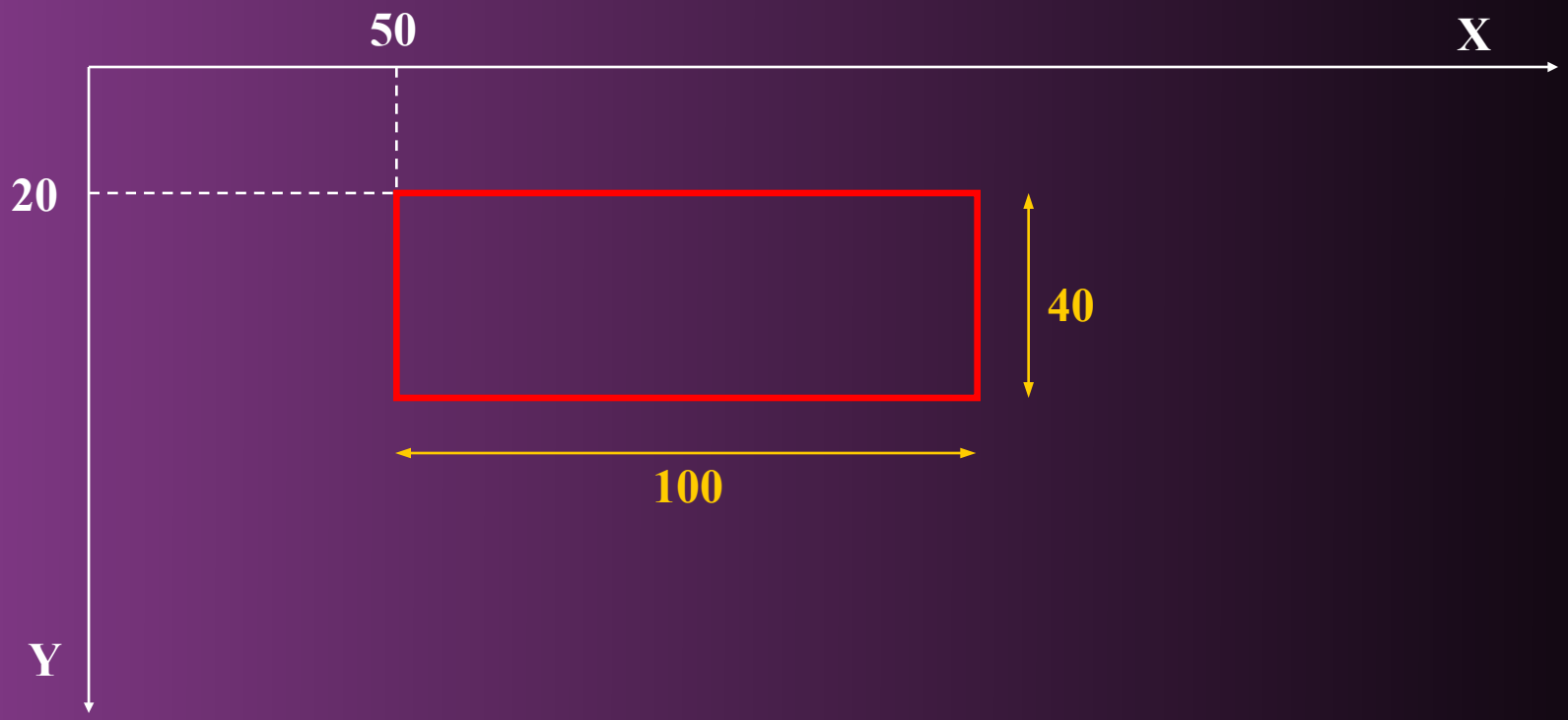
- The `drawRect` method draws the outline of a rectangle:

```
page.drawRect(x, y, width, height);
```

- The `fillRect` method draws a filled rectangle:

```
page.fillRect(x, y, width, height);
```

Drawing a Rectangle



page.drawRect (50, 20, 100, 40);

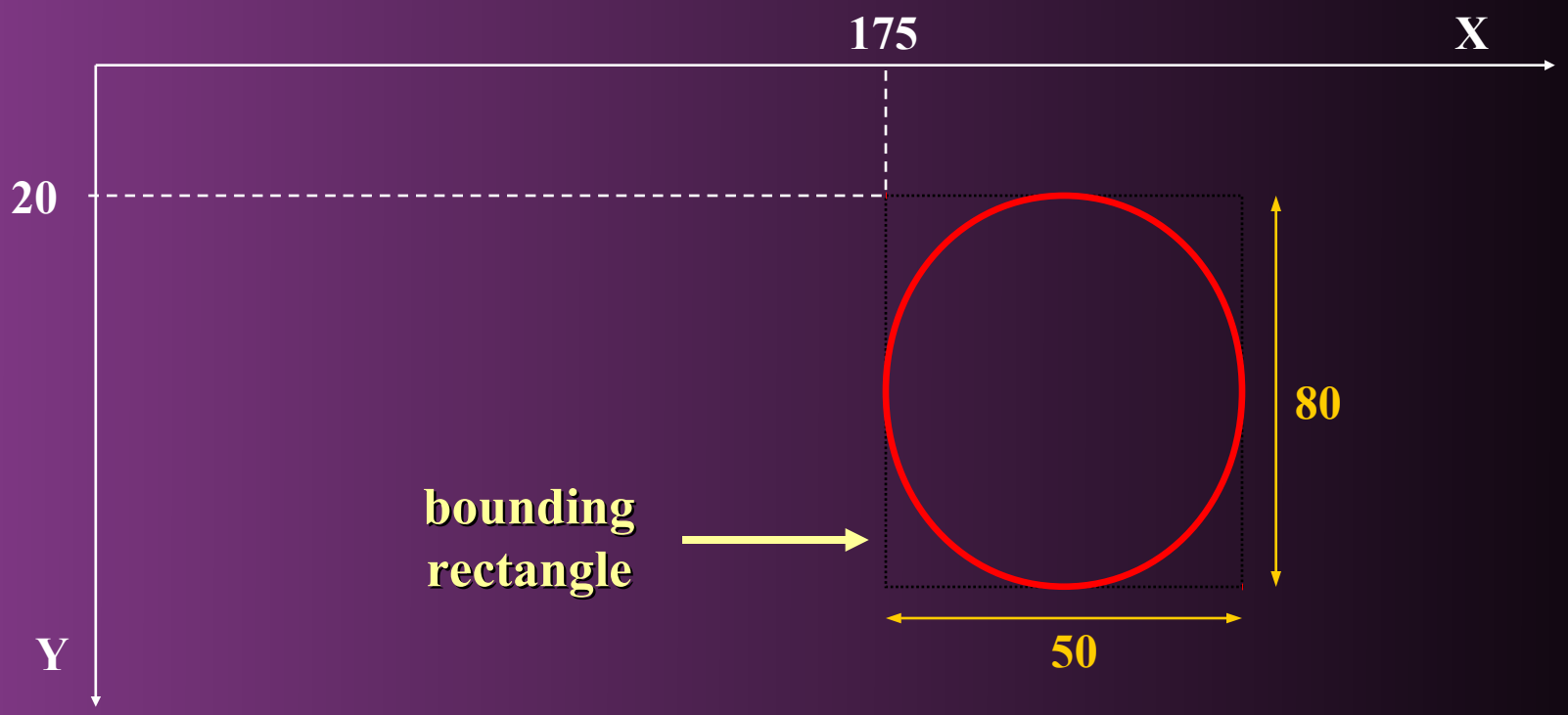
Ovals

- The `drawOval` and `fillOval` methods will draw the outline of an oval or a filled oval, respectively:

```
g.drawOval(x, y, width, height);
g.fillOval(x, y, width, height);
```

- `x` and `y` are the coordinates of the upper-left corner of an imaginary rectangle enclosing the oval.
- `width` and `height` are the measurements of this rectangle.

Drawing an Oval



page.drawOval (175, 20, 50, 80);

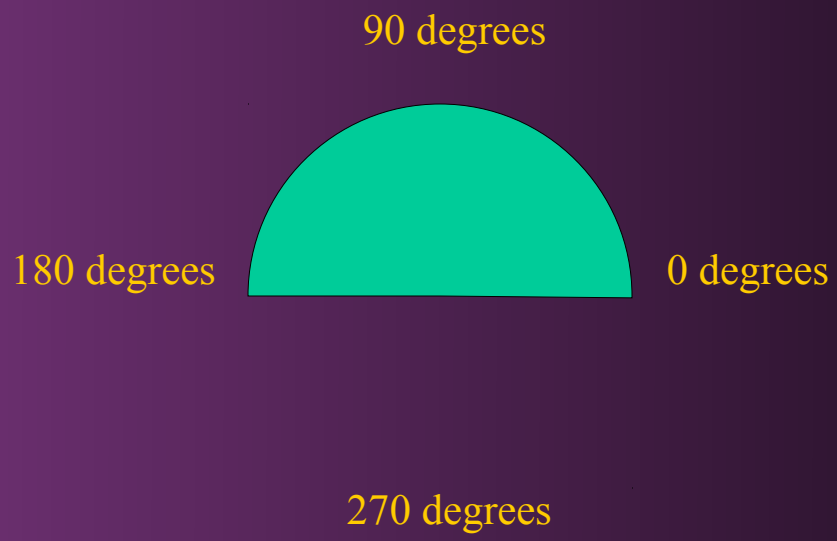
Drawing Arcs

- An arc is a segment of an oval.
- The `drawArc` method requires six arguments:

```
page.drawArc(x, y, width, height, startAngle, arcAngle);
```

- The last two arguments specify the angle at which the oval starts and the "arc angle" of the oval.
  - angles are measured in degrees, with zero degrees at 3 o'clock.
  - if the arc angle is positive, drawing is done in the counterclockwise direction.
  - if the arc angle is negative, drawing is done in the clockwise direction.

Arc positions

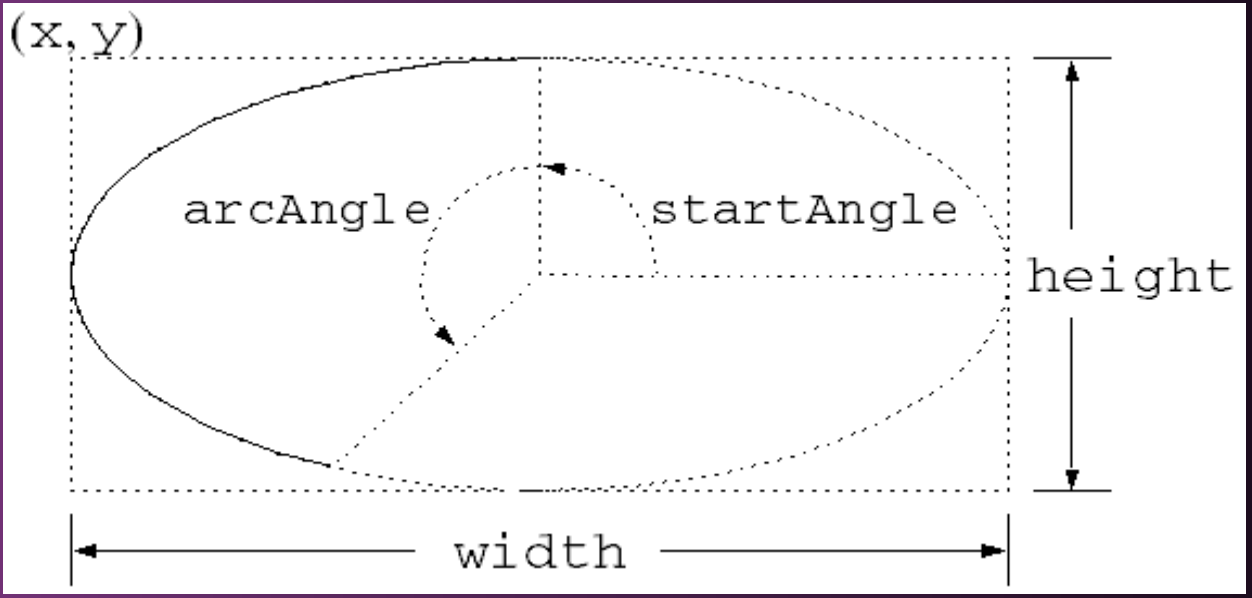


drawarc(x,y,w,h,0,180)
drawarc(x,y,w,h,180,-180)

} are identical

Drawing Arcs

- If `startAngle` is 90 and `arcAngle` is 135, `drawArc` will draw the following shape:

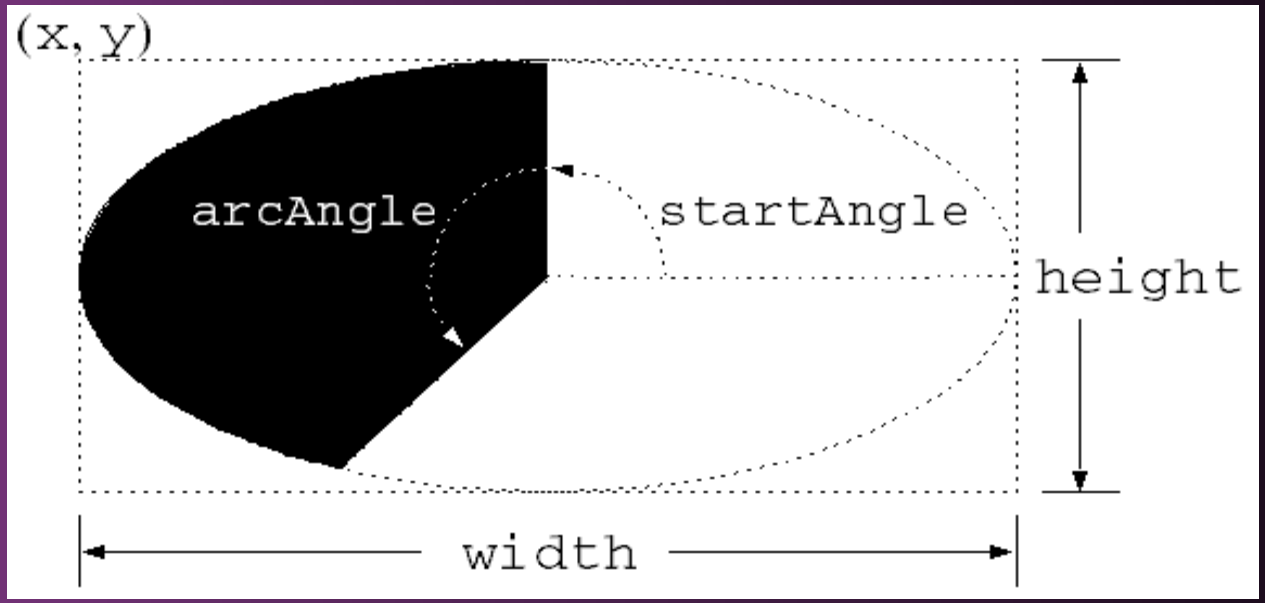


Filling Arcs

- The `fillArc` method draws a filled arc, which resembles a slice of pie.
- The arguments to `fillArc` are the same as those for `drawArc`.
- The filled area is a portion of the oval described by the first four arguments to `fillArc`.
- The values of `startAngle` and `arcAngle` determine which portion is filled.

Filling Arcs

- If `startAngle` is 90 and `arcAngle` is 135, `fillArc` will draw the following shape:



Drawing Polygons

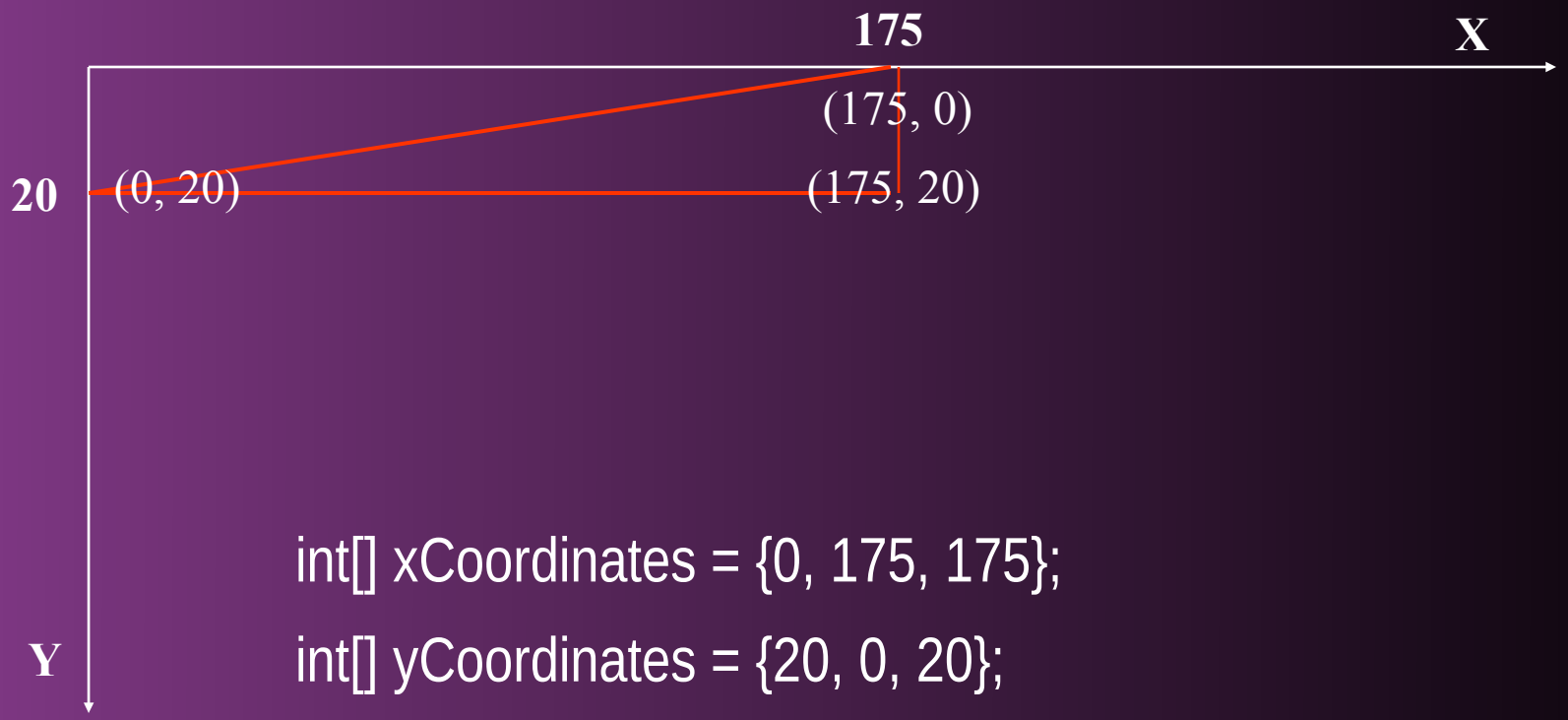
- A call of `drawPolygon` requires two arrays.
- One array contains the *x* coordinates of the polygon's vertices. The other array contains the *y* coordinates:

```
int[] xCoordinates = {0, 175, 175};
int[] yCoordinates = {20, 0, 20};
```

- `drawPolygon` also requires a third argument, indicating the number of vertices:

```
g.drawPolygon(xCoordinates, yCoordinates, xCoordinates.length);
```

Drawing a Polygon




```
int[] xCoordinates = {0, 175, 175};
int[] yCoordinates = {20, 0, 20};
```

```
page.drawPolygon (xCoordinates,  yCoordinates, xCoordinates.length);
```

```
page.fillPolygon (xCoordinates,  yCoordinates, xCoordinates.length);
```

Polylines

- A polyline is similar to a polygon except that its endpoints do not meet, and it cannot be filled
- There is also a separate `Polygon` class that can be used to define and draw a polygon

The Color Class

- The state of a `Graphics` object consists of more than just the pixels that it stores
- Each `Graphics` object also has a "current color" for drawing as well as a background color
- A color is defined in a Java program using an object created from the `Color` class of the `java.awt` package
- Each object of the class represents a single color

Foreground Color

- Every graphics context has a current foreground color
- All drawing is done in that color until the `setColor()` method of the `Graphics` class is called.
- By default, the drawing color is black.
- `getColor()` returns the current foreground color of the graphics context

Making colors

- A `Color` object can be created from three integers, indicating the red, green, and blue components of the color called the *RGB value*.
- Each component has a value between 0 (no contribution) and 255 (maximum contribution).
  - The color "black" has red, green, and blue values of 0
  - "white" has values of 255 for all three
- The Java API provides *constants* representing frequently used colors.
  - These constants are defined inside the `Color` class

Color Constants

Name	*Red Component*	*Green Component*	*Blue Component*
`black`	0	0	0
`blue`	0	0	255
`cyan`	0	255	255
`darkGray`	64	64	64
`gray`	128	128	128
`green`	0	255	0
`lightGray`	192	192	192
`magenta`	255	0	255
`orange`	255	200	0
`pink`	255	175	175
`red`	255	0	0
`white`	255	255	255
`yellow`	255	255	0

Calling the `setColor` Method

- A call of `setColor` that changes the current drawing color to the constant magenta:

```
public void paint (Graphics g)
{
        g.setColor(Color.magenta);
}
```

Calling the `setColor` Method

- New `Color` objects can also be created
- Code that sets the drawing color to pale blue:

```
Color paleBlue = new Color(64, 192, 255);
g.setColor(paleBlue);
```

Background Color

- Every drawing surface has a background color
- This is set with the `setBackground` method of the component on which we are working eg Applet

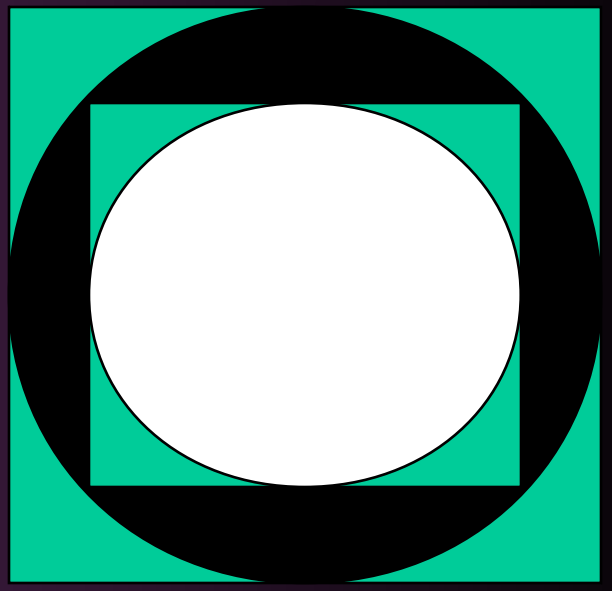
```
setBackground (Color.orange);
```

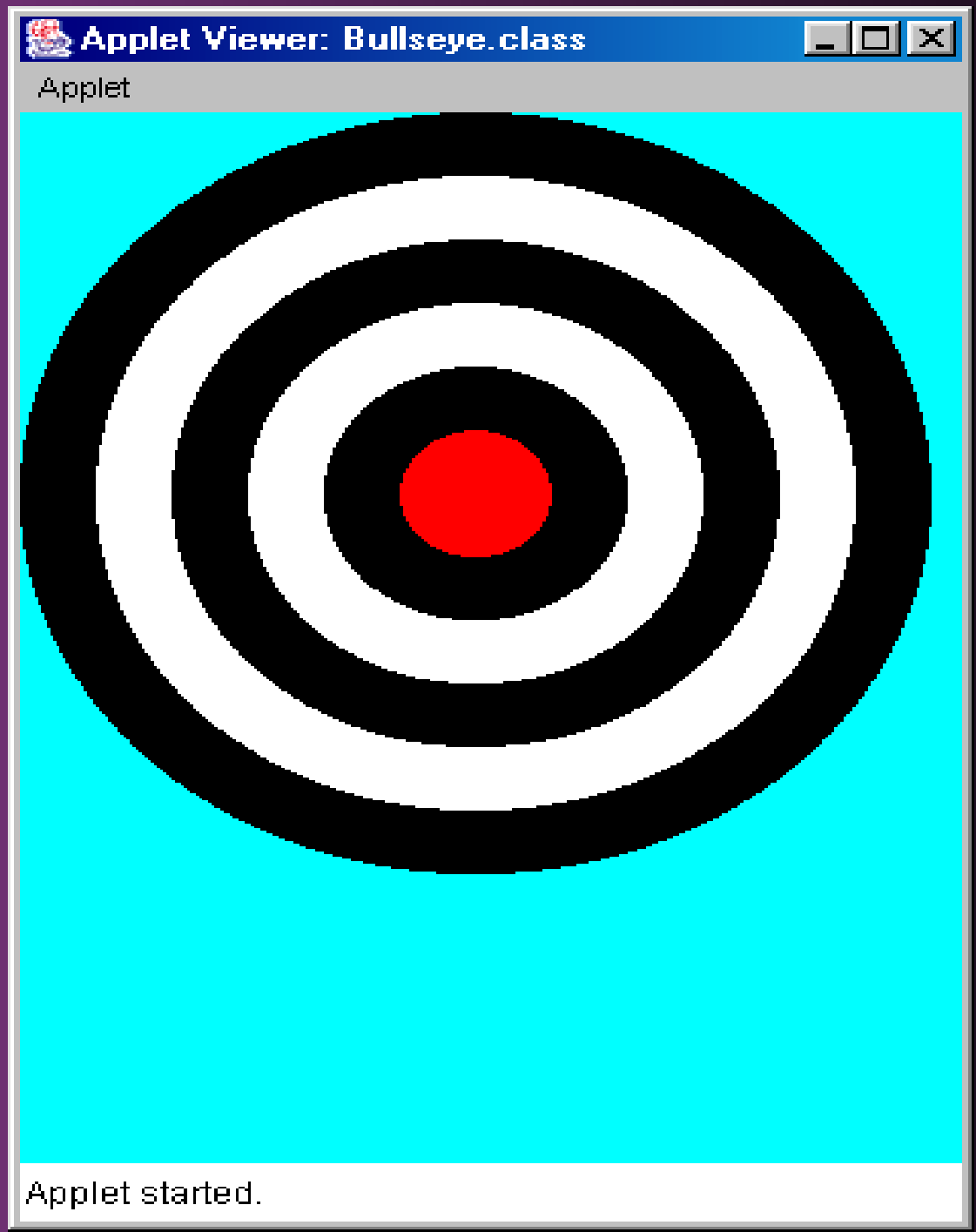
- Conditionals and loops can greatly enhance our ability to control graphics

```
import java.applet.Applet;
import java.awt.*;
public class Bullseye extends Applet       //colouring the bullseye omitted
{
   private final int MAX_WIDTH = 300;
   private final int NUM_RINGS = 5;
   private final int RING_WIDTH = 25;
   public void paint (Graphics page)
   {
      int x = 0, y = 0, diameter;
      setBackground (Color.cyan);
      diameter = MAX_WIDTH;
      page.setColor (Color.white);

      for (int count = 0; count < NUM_RINGS; count++)
      {
         if (page.getColor() == Color.black)  // alternate colors
            page.setColor (Color.white);
         else
            page.setColor (Color.black);

         page.fillOval (x, y, diameter, diameter);
         diameter = diameter - (2 * RING_WIDTH);
         x = x + RING_WIDTH;
         y = y + RING_WIDTH;
      }
   }
}
```




Applet Viewer: Bullseye.class
Applet
Applet started.

Displaying Text

- The `drawString` method is used to display text.

- `drawString` requires three arguments:
  - A string containing the text to be displayed.
  - The *x* and *y* coordinates at which the text is to be displayed.

- A call of `drawString`:

```
g.drawString("Java rules!", x, y);
```

Coordinates for Displaying Text

- `x` specifies the horizontal position of the first character in the string. `y` specifies the vertical position of the string's baseline:

The `Font` Class

- When text is written using `drawString`, the appearance of the text depends on the "current font," which is stored in every `Graphics` object.

- The current font can be changed by calling the `setFont` method:

```
g.setFont(newFont);
```

- The argument to `setFont` must be a `Font` object.

The `Font` Class

- `Font` objects are created by calling the `Font` constructor, which requires three arguments:
  - the font's name, style, and size.

- ***Font name.*** The font name is a string, such as `"Monospaced"`, `"Serif"`, or `"SansSerif"`.
- These three fonts are nearly universal
- Other font names are allowed, but there's no guarantee that a particular font will be available on the user's computer.

The `Font` Class

- ***Font style.*** Possible styles are bold, italic, and plain, represented by the constants `Font.BOLD`, `Font.ITALIC`, and `Font.PLAIN`.
- Writing `Font.BOLD + Font.ITALIC` creates a font that's both bold and italic.

- ***Font size.*** Font sizes are measured in points. (A point is approximately 1/72" or 0.35 mm.)

Calling the `setFont` Method

- An example of using the `Font` constructor:

```
Font f = new Font("SansSerif", Font.BOLD, 24);
```

- Passing `f` to the `setFont` method changes the current font:

```
g.setFont(f);
```

Another applet method

- Remember there is no `main` method in an applet

public void init()

- This method is executed once when the applet is first loaded

- Use it to:
  - Eg Get data from a user to use in the paint method

Using the method init()

```
public class DrawLines extends Applet
{
    // why might you define certain variables here?

    public void init()
    {
        // eg get data from user
    }

    public void paint(Graphics g)
    {
        // draw shapes etc
    }
}
```

Loops and Exceptions Practice

- Write code to prompt a user to enter an integer
- Include exception handling for non-integer data entry
  - Display "You must enter an integer");
- Use a *while* loop to repeat if non-integer data entered controlled by a *boolean*

```
boolean invalidInput = true;


 while (invalidInput)     //repeats if non-integer data entered
{
     intStr = JOptionPane.showInputDialog (“Enter an integer");
     try
     {
          intNum = Integer.parseInt(intStr);
          invalidInput = false; // sentinel set to stop loop if integer entered
     }

     catch (NumberFormatException e)
     {
          JOptionPane.showMessageDialog (null, "You must enter an integer");
     }
} //end while
 invalidInput = true; // reset sentinel after valid data entered
                      // in case user wants to loop around the whole program
```

Modify the program

- You want to enter 3 integers – add a *for* loop
- The integers must be in the range 1-10
- If valid, store the integer in an array called intArray

```
int num = 0;
int [] intArray = new int [3];

for (index = 0; index < intArray.length; index ++)
{
       while (num < 1 || num > 10)     //repeats if non-integer data entered
       {
            numStr = JOptionPane.showInputDialog (“Enter an integer in range 1-10");
            try
            {
                 num = Integer.parseInt(numStr);
            }

            catch (NumberFormatException e)
            {
                  JOptionPane.showMessageDialog (null, "You must enter an integer");
            }
       } //end while
       intArray[index] = num;
       num = 0;  //reset num so while loop will execute to get next 2 numbers

} //end for
```

Lecture Outcomes

Today we have covered:

- Introduction to applets
- Graphical methods

- Questions?