IS311 Programming Concepts

Abstract Window Toolkit
(part 1: Drawing Simple Graphics)

Abstract Window Toolkit

The Abstract Window Toolkit (AWT) package contains all the classes for creating user interfaces and for painting graphics and images.

A user interface object such as a button or scrollbar in AWT terminology is called a ***component***.

Graphics

คลาสพื้นฐานสำหรับการเขียนภาพกราฟฟิก

Graphics	เป็นคลาสที่ใช้สำหรับการวาดภาพกราฟฟิกเช่นลากเส้น เขียนรูปทรงต่าง ๆ
Color	Represent a color
Font	Represent a font
FontMetrics	Used for determining information about a font
Image	Represent an image

The Java Graphics System

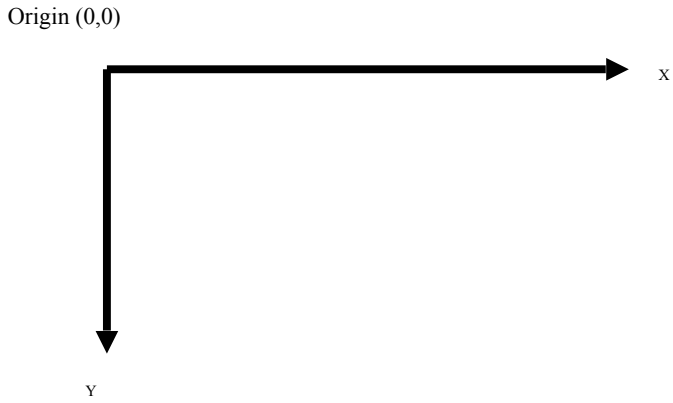
Java provides a set of graphic commands that allow programmer to:

- Display graphical shapes on the screen
  - size shape location are under programmers control
- Display strings
  - size, font, style are under programmers control
- Display images
- Color these objects
- Move these objects

Coordinate Systems

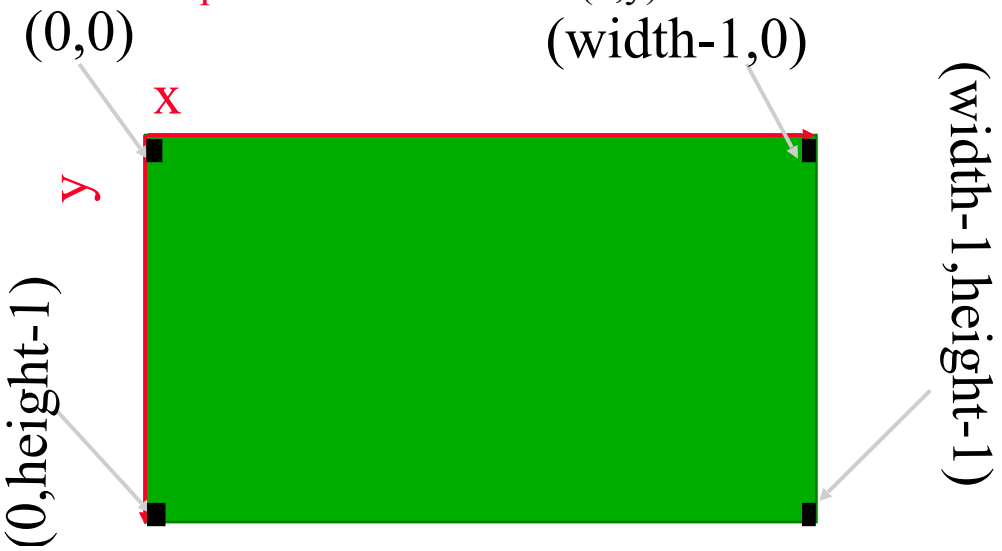
- Java's coordinate system is not like the coordinate system you will use in physics and general math classes
- The Origin is in the upper left hand corner
- X increases to the right
- Y increases downward
- The dimensional unit is a pixel

Java Graphic Coordinate System



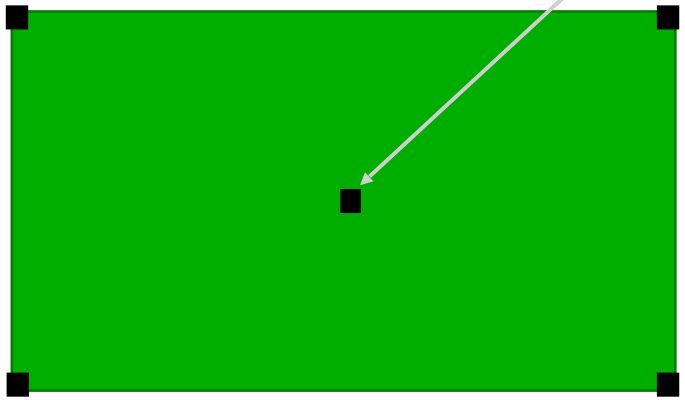
Java Graphic Coordinate System

Each pixel has a coordinate (x,y)

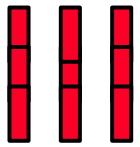


EXERCISE

Let `width` and `height` be odd. What are the coordinates of the *middle pixel*?



SOLUTION



```
width = 3, height = 3
answer = (1,1)
```

```
answer = ((width-1)/2,(height-1)/2)
```

ถ้าความกว้าง และความสูงมีค่าเป็นเลขคู่ละ จะคำนวณอย่างไร?

$$answer = \left(\lfloor width/2 \rfloor, \lfloor height/2 \rfloor\right)$$

```
answer = (width/2,height/2)
```

Geometry

Dimension	Used for specifying the size of a rectangle (width and height)
Insets	Used for specifying the insets of a rectangle (top, left, bottom, and right)
Point	Used for specifying a point x, y coordinates.
Polygon	Used for holding an array of points.
Rectangle	Used for specifying the location and size of a rectangle (x, y; width; and height).

Drawn and Filled Shapes

- Java lets you draw lines and shapes
- Java shape drawing methods come in two styles
  - those where the outline only is shown
    - **draw*Shape*()** **draw(*shapeClass*)**
  - those where the center is filled
    - **fill*Shape*()** **fill(*shapeClass*)**
- Java provides the ability to display predefined images
- Java provides the ability to display widgets

Displaying Things

- First we need a Graphics context
  - That portion of the screen to draw upon
- How do we obtain a graphics context?
  - We get one as the argument to **`paint()`**
    - Be careful about passing the graphics context around,
      - it may not always be 'in context'
- Java is state driven
  - Things tend to stay they way they are until they are changed
    - Once we set a color it will stay that way

Some Graphical things

- Line segments (เส้นตรง)
- Connected line segments (ลากเส้นตรงหลายเส้นต่อเนื่องกัน)
- Rectangles (รูปสี่เหลี่ยม)
- Ellipse (รูปวงรี)
- Arcs (เส้นโค้ง)
- Rectangles with rounded corners (สี่เหลี่ยมมุมมล)
- Polygon (รูปหลายเหลี่ยม)

Drawing Lines

- The 1.1 AWT provides three methods for drawing lines:
  - Straight Line Segments
    `drawLine(...)`
  - Connected Line Segments
    `drawPolyline(...)`
  - Portions of an Ellipse
    `drawArc(...)`

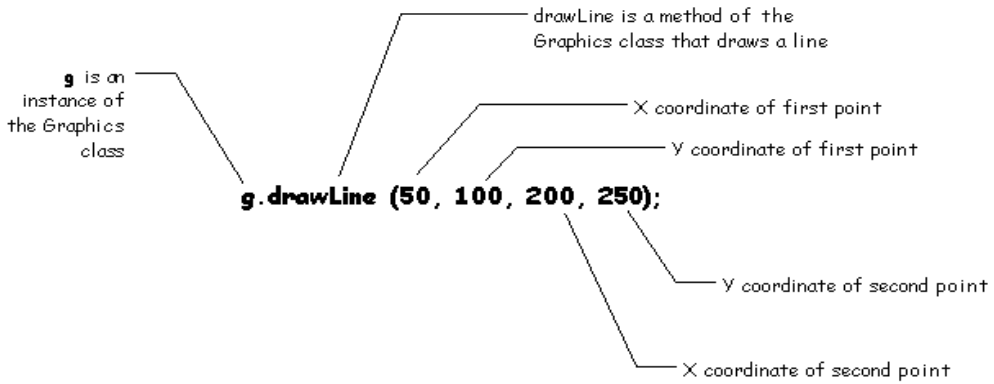
Line Segments

- The command is of the form:

```
GraphicsObject.drawLine(int xStart, int
  yStart, int xStop, int yStop)
```

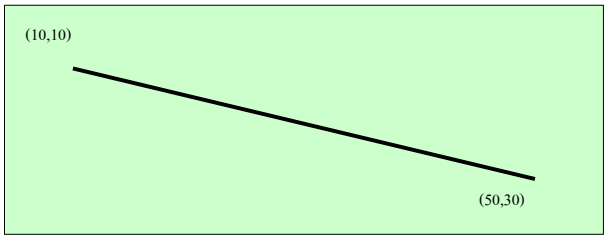
- Draws a line from position point (`xStart, yStart`) to (`xStop, yStop`)

The **`drawLine`** method



Example of **drawLine()**

```
public void paint( Graphics g ){
  g.drawLine( 10, 10, 50, 30 );
}
```



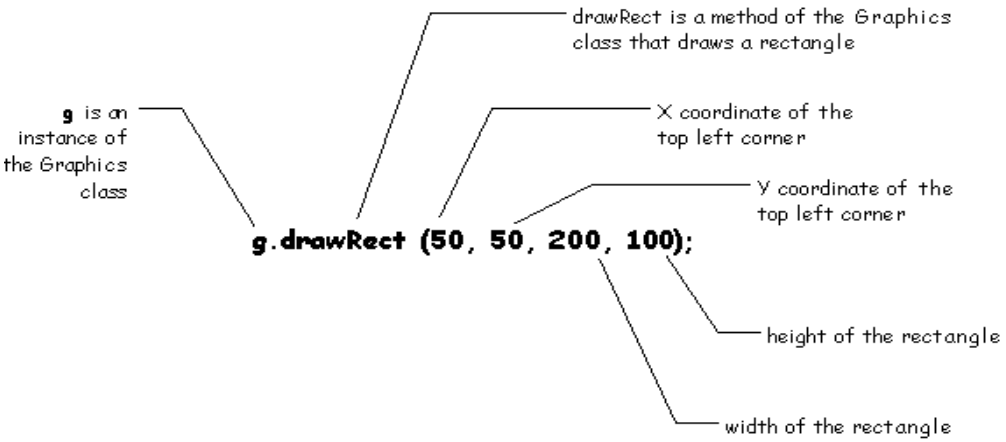
Drawing and Filling Rectangles

- **Drawing Rectangles**

```
void drawRect(int x,      // upper left x coordinate
              int y,      // upper left y coordinate
              int width,  // bounding box width
              int height) // bounding box height
```

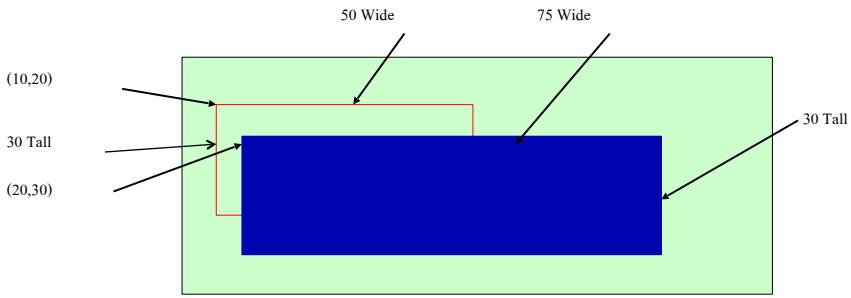
- Draws an outline of a rectangle bounded whose upper left hand corner is defined at the point (`x,y`), and is `width` wide, and `height` tall.
- To draw a solid rectangle we use the **`fillRect()`** method with the same arguments

The **drawRect** Method



Example Rectangles

```
public void paint(Graphics g) {
  g.setColor(Color.red);
  g.drawRect(10,20,50,30);
  g.setColor(Color.blue);
  g.fillRect(20,30,75,30); ← fillRect เติมสีลงในสี่เหลี่ยม กำหนด
                           พารามิเตอร์เช่นเดียวกับ drawRect
}
```



Rectangles with Rounded Corners

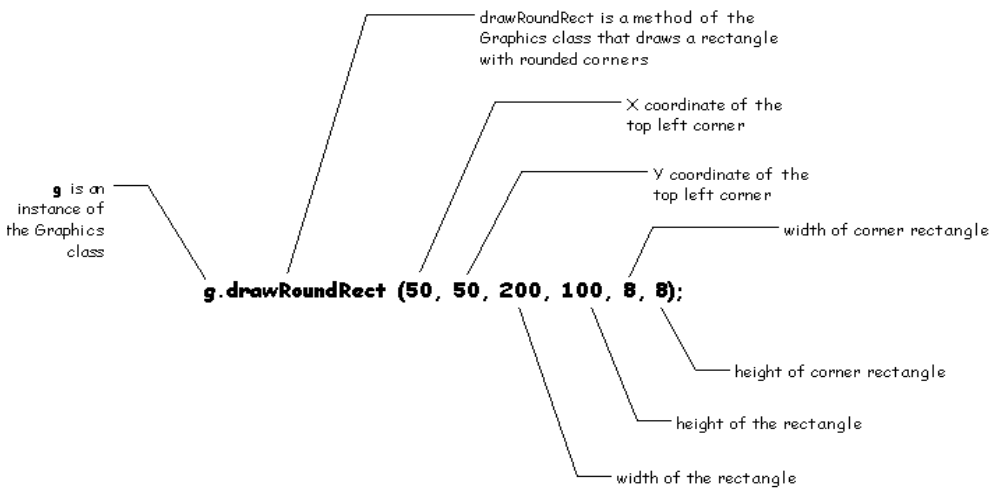
- We can combine our approach for drawing rectangles and drawing ellipses into a a shape which is a rectangle with rounded corners or curved ends.
- The Java API command `drawRoundRect()` combines the complexity of rectangles with the difficulty of ellipses.

The **`drawRoundRect`** method

```
void drawRoundRect(int x,          // First four
                   int y,          // parameters
                   int width, //are as in a
                   int height,// rectangle

                   int arcWidth,
                   int arcHeight)
                      //horizontal and vertical
                      // 'diameter' at the
                      // at the corners
```


The **`drawRoundRect`** method

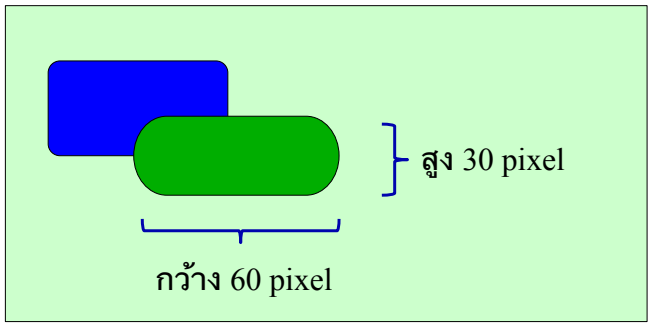


More rounded rectangle stuff

- The first four parameters are exactly the same as those for the rectangle.
  - If we put zeros in for the last two parameters, you will have a rectangle drawn exactly as before.
- We use the fifth argument in the `drawRoundRect()` method as we do the third argument in the `drawOval()` method.
  - This describes the horizontal 'diameter' of the arc.
- The sixth argument in the `drawRoundRect()` method corresponds to the fourth argument in the `drawOval()` method.
  - This describes the vertical 'diameter' of the arc.
- Of course, there is a `fillRoundRect()` for filling in our rectangle with a solid color.

Example Rounded Rectangles

```
public void paint(Graphics g) {
  g.setColor(Color.blue);
  g.fillRoundRect(10,20,60,30,5,5);
  g.setColor(Color.green);
  g.fillRoundRect(40,30,60,30,15,15);
}
```

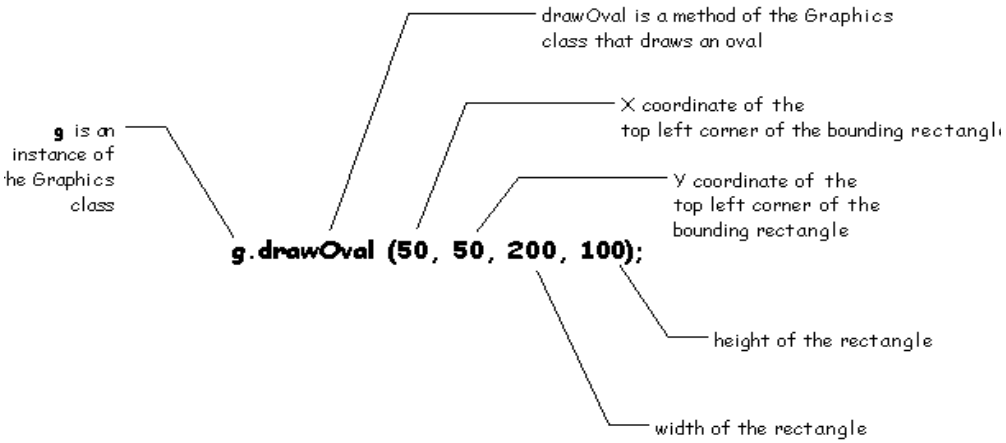


Drawing Ellipses

```
void drawOval(int x,      // upper left x coordinate
              int y,      // upper left y coordinate
              int width,  // bounding box width
              int height) // bounding box height
```

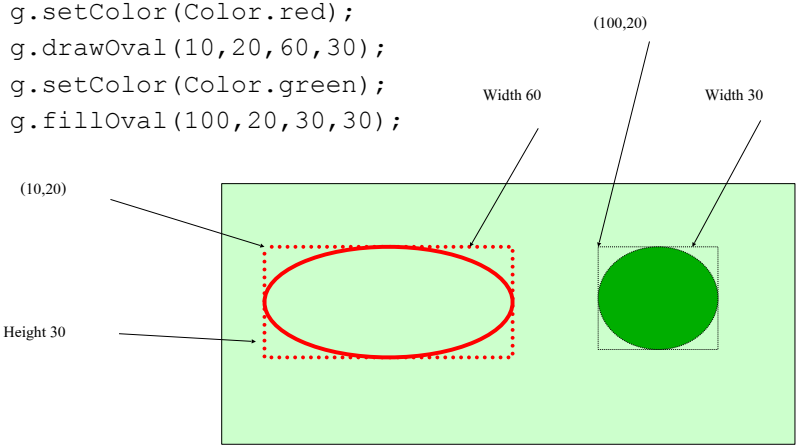
- This draws an outline of an ellipse bounded by the rectangle whose upper left hand corner is defined at the point `(x,y)`, and is `width` wide, and `height` tall.
  - Note that the point `(x,y)` does not actually fall on our arc.

The **`drawOval`** method



Example Ellipses

```
public void paint(Graphics g) {
  g.setColor(Color.red);
  g.drawOval(10,20,60,30);
  g.setColor(Color.green);
  g.fillOval(100,20,30,30);
}
```



Circles, & Fills

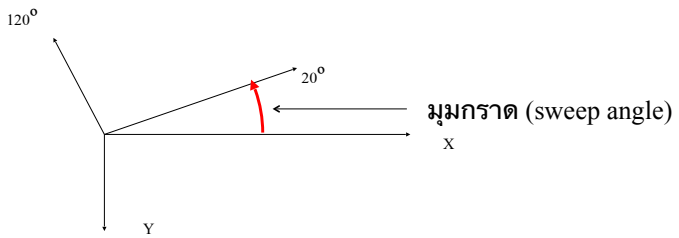
- The JDK does not provide a method to draw a circle. To do that we must set our bounding box to be a square, (the width and the height are the same size).
- To draw a solid, we use the `fillOval()` method. The `fillOval()` takes exactly the same arguments as the `drawOval()`.

Drawing Arcs

- We can draw smooth curves as a portion of an ellipse
- The **`drawArc()`** command draws an arc bounded by the rectangle whose upper left hand corner is defined at the point `(x,y)`, and is `width` wide, and `height` tall.
- If we were to draw a complete circle, it would touch the centers (midpoints) of each of the sides of our bonding box.
  - However, we define our curve to be drawn from *startAngle* to an arc of *sweepAngle* degrees.

Angle Measurements in Java

- Angles are measured counter clockwise from the horizontal x axis.
  - In Java, 0 degrees is at the 3-o'clock position.
  - Positive angles indicate counter-clockwise rotations, negative angles are drawn clockwise.

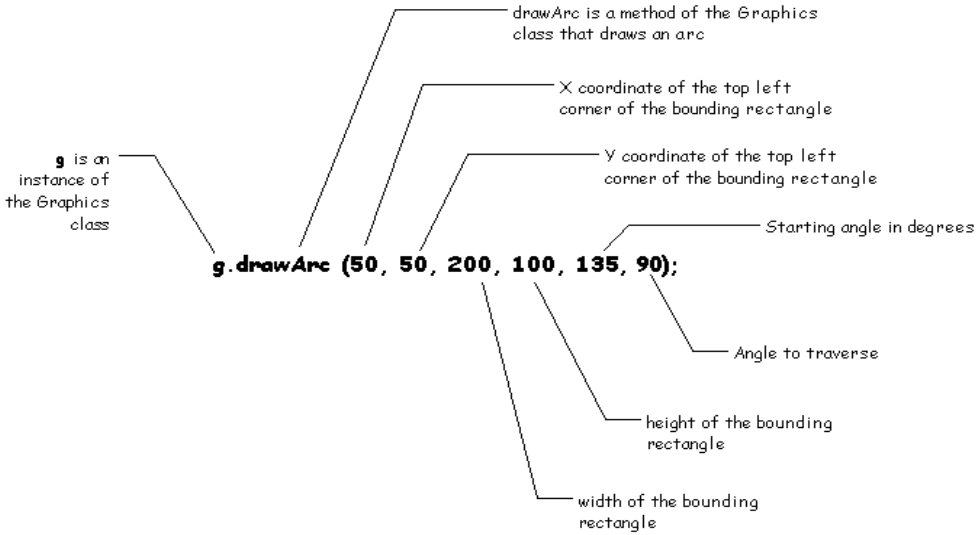


The **`drawArc`** method

```
void drawArc(int x,       // bounding box, x
                          // upper left corner
             int y,       // bounding box, y
                          // upper left corner
             int width,       // width of
                              // bounding box
             int height,      // height of
                              // bounding box
             int startAngle,  // in degrees
             int sweepAngle)  // extent
```

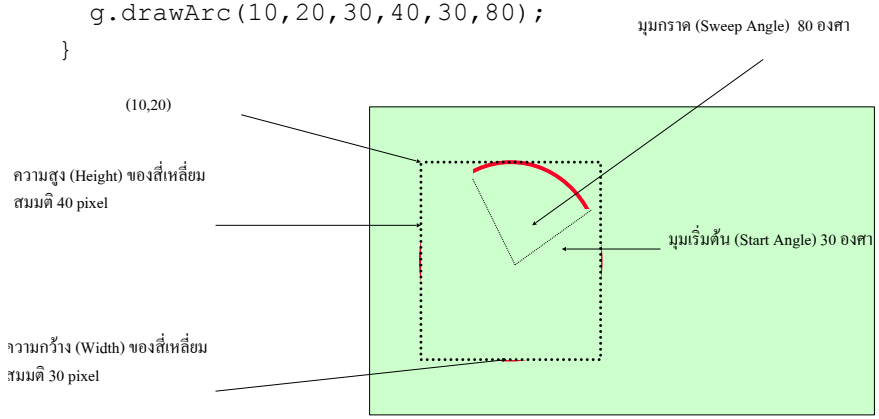
สามารถใช้ **`fillArc()`** เพื่อระบายเสี้ยวของวงรีได้

The **drawArc** method



Example drawArc()

```
public void paint(Graphics g) {
  g.drawArc(10,20,30,40,30,80);
}
```



The **drawPolyline** method

```
drawPolyline(
 int[] xPoints, // array of x values
 int[] yPoints, // array of y values
 int nPoints  // number of points
)
```

example

```
int [] xPoints = { 1,2,3,4, 5, 6, 7, 8, 9};
int [] yPoints = { 3,4,5,7,10,15,20,27,35};

g.drawPolyline(xPoints, yPoints, xPoints.length);
```


Polygons

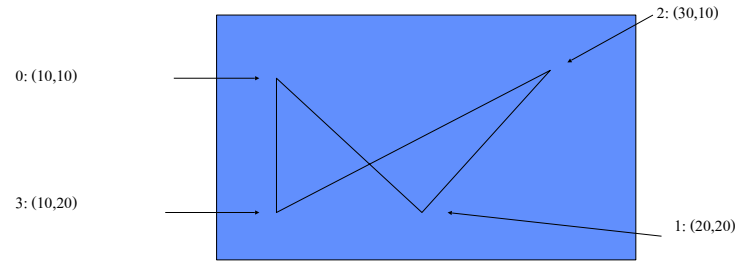
- Like the **polyline** method, this accepts an array of x's and y's.
  - However, unlike the **polyline** method, the polygon will create the line that *connects the last point in polygon to the first point in the polygon.*
- As in the **polyline**, you must be certain that the number of points specified is equal to or smaller than the smallest number of points in ether of the arrays.

drawPolygon()

- One form of the drawpoly command:
  `void drawPolygon(Polygon p)`
- Uses a polygon object:
  - We must first create a polygon object: (p, above).
  - To do this we would declare and instantiate a polygon:
    `Polygon p = new Polygon();`
  - We then add points to the polygon:
    `p.addPoint(x,y);`
    `p.translate(dx, dy);` ย้ายตำแหน่ง

Example Polygon

```
public void paint(Graphics g) {
  Polygon poly = new Polygon();
  poly.addPoint(10,10); poly.addPoint(20,20);
  poly.addPoint(30,10); poly.addPoint(10,20);
  g.drawPolygon(poly);
}
```



The drawString method

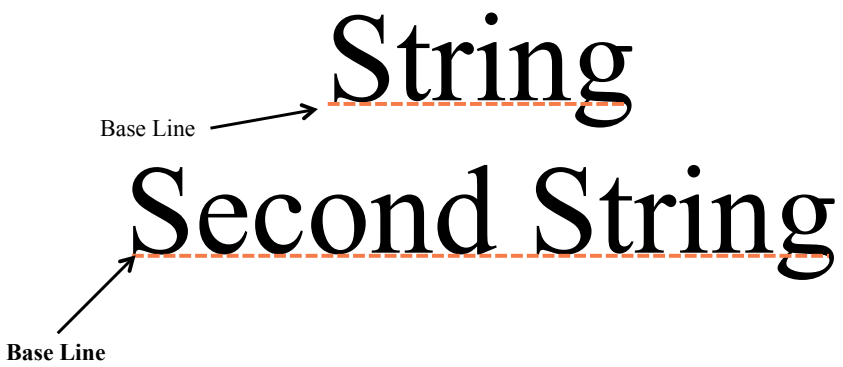
```
void drawString(
          String s, // string to be drawn
               int x,    // base line x coordinate
          int y          // base line y coordinate
)
```

```
g.drawString("Hello World!", 10, 10);
```


Fonts and Strings

- Java provides a flexible system for displaying strings
  - We can change the type face
  - We can change the size
  - We can change the weight or style

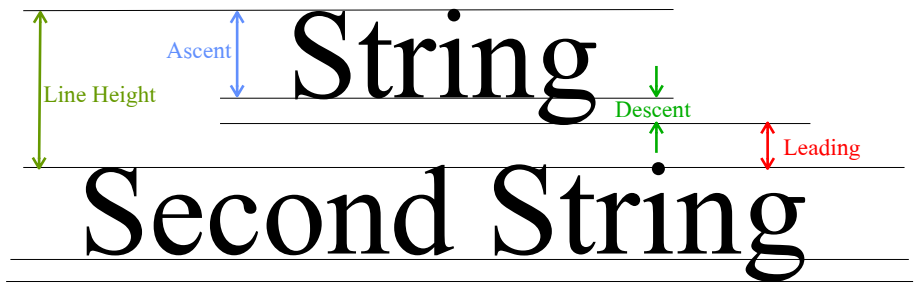
Consider the Following



Baseline Observations

- Notice that nearly all the letters do not extend below an imaginary line
  - except the letter 'g'?
- This imaginary line is called the base line.
- Our reference (anchor) point for a string in the y dimension is this base line.
  - Unlike any of the other graphics objects,
- In the x direction, we start the string on the left hand side.

Font Dimensions



Line Height = Ascent + Descent + Leading

Font Definitions: Ascent

- **font ascent:** This is the distance from the base line to the top of the letter.
  - In our example above, this is the distance from bottom to the top of the letter 'S'.
  - When we want to know about a specific font, we're probably interested in the *maximum* height for all characters we can display in this font, this is provided by the method `getMaxAscent()`.

Font Definitions: Decent

- **font descent:** This is the distance below the line to the bottom of the letter.
  - In our example above, this is the distance from the bottom of the 'S' to the bottom of the 'g'.
  - As with the ascent, we're probably more interested in the Maximum Descent.

    `getMaxDescent()`

Definitions: Leading & Height

- **leading:** The distance between base lines in strings is called the leading distance.
  - In our example above, this is the distance between the bottom of the 'g' and the top of the 'S' below.
- **line height:** The sum of these three distances.

The Use of the Leading dimension

- We are not required to use the leading and font height dimensions to place our strings on the screen.
- You may have a very good reason to use a different dimension for leading that what is provided to you .
  - You might want a double space effect.
- However, these are dimensions that are 'recommended by the experts.'

Doing things with fonts

- We can set fonts

  `graphicsObject``.setFont( Font f )`
- We can find out what the current font is

  `graphicsObject``.getFont()`
- We can use the font

  `graphicsObject``.drawString(`
  `String s, int x, int y)`
- All of these need a graphics object
  - Such as that provided with `paint()`

Setting Fonts

- To set a font we need a Font object
- To create a Font object we will create one like any other Java object

```
new Font (String fontName,
  int fontStyle, int fontSize)
```

  - The Font Name is one of several predefined constants
  - The Font Style is selected from 3 constants
  - The Font Size is (roughly) the number of pixels used for the font height

Font Names

- Java 1.1(Java 1.0)
  - Monospaced (Courier), Dialog, Dialog Input, SanSerif(Helvetica), Serif (Times Roman), Symbol
- MS Windows
  - `Courier New`, MS San Serif, MS San Serif , Arial, Times New Roman, WingDings
- Macintosh
  - `Courier`, Geneva, Geneva, Helvetica, Times Roman, Symbol

Font Styles

- In the setFont method, we can set font styles:
  `Font.PLAIN`
- **Bold**
  `Font.BOLD`
- *Italics*
  `Font.ITALIC`
- ***Combination***
  `Font.BOLD + Font.ITALIC`

Obtaining Font Dimensions

To get the detailed dimensions about a font we'll get a Font Metrics object.

- (Need a current Graphics object)

```
FontMetrics FM_object =
  graphics_object.getFontMetrics(Font f);
```

Working with Font Metrics

- We can then get our maximum height above the base line
  `FM_object.getMaxAscent()`
- We can get the maximum descent below the base line with the following method.
  `FM_objet.getMaxDescent()`
- We can get line hight with the following method.
  `FM_object.getHight()`

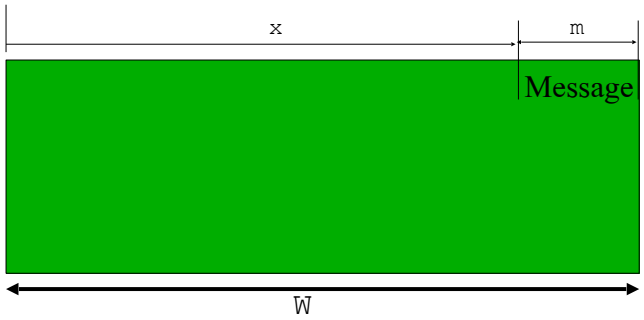
Finding the Width of a String

- To find the width of a string we use the **`FontMetrics`** method **`stringWidth`**.
  - We pass this method the string we want to find the length for as its argument.

```
FM_object.stringWidth( myString )
FM_object.stringWidth("Static String")
```

ตัวอย่างจัดแสดงสตริงชิดขวา

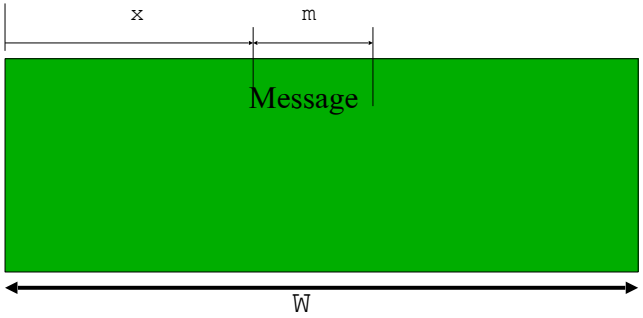
ถ้ามีสตริงความยาว *m* pixels ต้องการแสดงชิดด้านขวาของวินโดว์หรือ applet ซึ่งมีความกว้าง *w* pixels จะคำนวณตำแหน่งของ *x* ที่จะ `drawString` ได้อย่างไร



คำนวณหาค่า x ได้จาก `x = w - m`

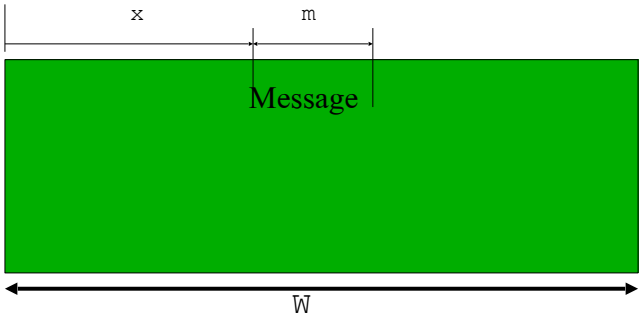
EXERCISE

ถ้ามีสตริงความยาว *m* pixels ต้องการแสดงไว้กลางวินโดว์หรือ applet ซึ่งมีความกว้าง *w* pixels จะคำนวณตำแหน่งของ *x* ที่จะ `drawString` ได้อย่างไร



SOLUTION

Centering a String



Answer

```
x = ( w - m ) / 2
```

String Centering Details

- Assume you have the following
  - Window width, `w` // version dependent
  - A String, `msg`
  - Graphics object, `g`
  - Font object, `f`

```
FontMetrics fm = g.getFontMetrics( f );
x = ( w - fm.stringWidth(msg) ) / 2;
g.drawString(msg, x, y);
```

Color

Java gives us control over the color we draw things in

- We set the color

  `graphicsContext.setColor(Color c)`
- We can use the pre-defined colors

  `Color c = Color.colorName`
- We can create our own

  `Color c = new Color(...);`

The pre-defined colors

black, blue, cyan, gray, dark gray, light gray, green, magenta orange, pink, red, white, yellow

Creating our own colors

- On computers, we generally create colors using three components of light.
  - Our component colors are Red, Green and Blue.
    - Note this is not the same as with pigment colors, where the primary colors are Red, Blue and Yellow
- Using our light color model:
  - Black is the absence of any of the three colors.
  - White is the presence of all three at full intensity

The Three Primary Colors

	Red Light	Green Light	Blue Light
Red Color	100%	0%	0%
Green Color	0%	100%	0%
Blue Color	0%	0%	100%

Variations of the Primary Colors

	Red Light	Green Light	Blue Light
Medium Red Color	75%	0%	0%
Dark Green Color	0%	50%	0%
Navy Blue Color	0%	0%	25%

- All other colors can be obtained by are combinations of the these basic three colors in various proportions and intensities.
- Gray is an equal proportion of all three colors at an intensity somewhere between white (100%) and black (0%).

Specifying New Colors

```
Color myColor = new Color( ... )
```

- Three forms:

```
Color( float red_fraction,
       float green_fraction,
       float blue_fraction)
```

where the fraction is from 0.0 to 1.0

```
Color( int red_part, int green_part,
     int blue_part)
```

where the values are from 0 to 255

```
Color(int RGBvalue)
```

RGBvalue = red_part * 256 * 256 + green_part*256 + blue_part

Changing Colors

```
Color.darker()
Color.brighter()
```

- We can even combine these:

```
Color.darker().darker().darker();
Color.darker().brighter()
```


Java Applets

Java *applets* provide for client-side programming

- compiled into Java byte code, then *downloaded as part of a Web page*
- executed by the JVM *embedded within the Web browser*
- unlike JavaScript, Java is full-featured with extensive library support
- Java and its APIs have become industry standards
  - ➢the language definition is controlled by Sun, ensures compatibility
  - ➢Applications Programming Interfaces standardize the behavior of useful classes and libraries of routines

Java Applets (cont.)

important point: Java applets & applications look different!

- if you want to define a stand-alone application, make an application requires `public static void main` method
- if you want to embed the code in a Web page, make an applet requires `public void paint`, `public void init`, ...
- can define dual-purpose programs, but tricky

First Java applet

```
import java.awt.*;
import java.applet.*;
/**
 * This class displays "Hello world!" on the applet window.
 */
public class HelloWorld extends Applet
{
  public void paint(Graphics g)
  {
     g.drawString("Hello world!", 10, 10); // writes string at (10,10)
  }
}
```

libraries: Java provides extensive library support in the form of classes

- libraries are loaded using import
  - `java.awt`: contains Abstract Window Toolkit (for GUI classes & routines)
  - `java.applet`: contains the applet class definition

First Java applet

```
import java.awt.*;
import java.applet.*;
/**
 * This class displays "Hello world!" on the applet window.
 */
public class HelloWorld extends Applet
{
  public void paint(Graphics g)
  {
     g.drawString("Hello world!", 10, 10); // writes string at (10,10)
  }
}
```

all applets inherit from the Applet class (in **`java.applet`**)

default methods include:

- `init()`: called when page is loaded to create/initialize variables
  by default, does nothing
- `paint(Graphics g)`: called to draw (after init) or redraw (after being obscured)
  here, the paint method is overridden to display text on the applet window

Embedding an applet in HTML

to include an applet in a Web page, use either

- APPLET tag (deprecated)
  CODE specifies applet name, HEIGHT and WIDTH specify window size
  text between the APPLET tags is displayed if unable to execute (e.g., Java not enabled)
- OBJECT tag
  preferred for HTML 4, but not universally supported

```
<html>
<head>
  <title>Hello World Page</title>
</head>
<body>
  <applet code="HelloWorld.class" height="100" width="100">
    You must use a Java-enabled browser to view this applet.
  </applet>
</body>
</html>
```

view page in browser at http://is311.bus.tu.ac.th/examples/applet/ds1.html

Applet parameters

```
import java.awt.*;
import java.applet.*;
// This class displays a message based on parameters.
public class HelloWorld1 extends Applet
{
    public void paint(Graphics g)
    {
       String userName = getParameter("name");
        int userAge = Integer.parseInt(getParameter("age"));
        String message1 = "Hello " + userName + ".";
       String message2 = "On your next birthday, you will be " +
                          (userAge+1) + " years old.";
       g.drawString(message1, 10, 10);
       g.drawString(message2, 10, 30);
    }
}
```

can access parameters passed in from the HTML document

`getParameter` accesses the value of the parameter (must know its name)

- if the parameter represents a number, must `parseInt` or `parseFloat`

Parameters in HTML

```
<html>
<head>
<title>Hello World with parameters Page</title>
</head>
<body>
<applet code="HelloWorld1.class" height=35 width=300>
  <param name="name" value="Chris">
  <param name="age" value="20">
  You must use a Java-enabled browser to view this applet.
</applet>
</body>
</html>
```

can specify parameters to the APPLET when it is embedded in HTML

- each parameter must have its own PARAM tag inside the APPLET element
- specifies parameter name and value

view page in browser at http://is311.bus.tu.ac.th/examples/applet/ds2.html