CSE1720

Week 04, Lecture 07

Click to edit Master text styles
Second level
Third level
Fifth level

Winter 2014 ◆ Tuesday, January 28, 2013

YORK UNIVERSITÉ UNIVERSITY

Objectives for this class meeting

- Cover basic information on 2D Graphics
  - we will need this for designing our game

Basic Graphics

- Background Material:
- sec 8.1.5, lab L8.2 (pp.329-332).
- The Java Tutorials, **Trail: 2D Graphics**
- **http://docs.oracle.com/javase/tutorial/2d/index.html**
- These lecture slides provide a basic overview of that material, enough to get you started with the lab exercises

The Big Picture

- apps that use graphics must work with the **Window Manager (WM)**
- in order to understand how to use graphics, you should have a basic understanding of the WM

The Window Manager

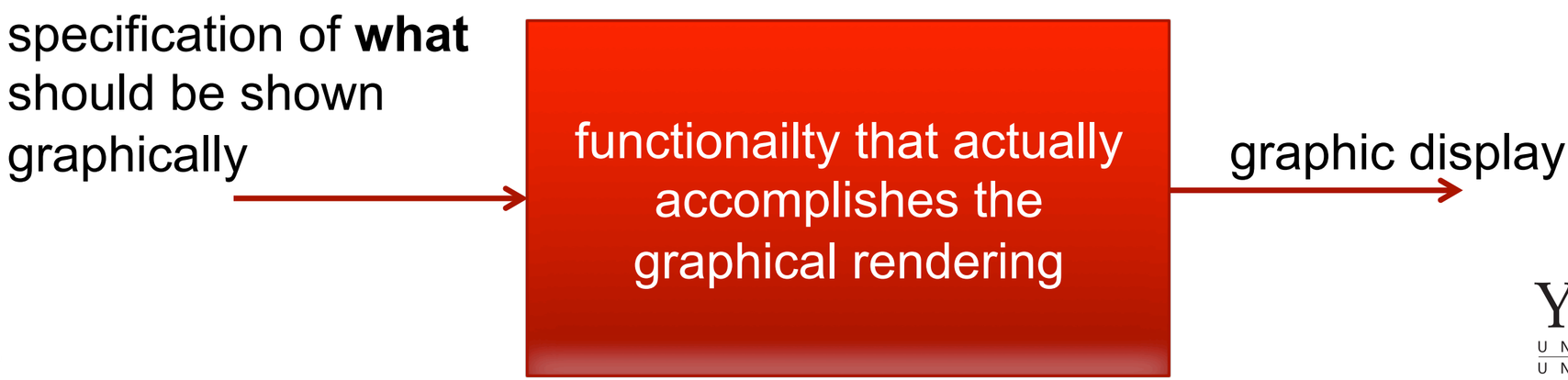
- The WM is used to implement GUI-based user interfaces
- The WM is part of the operating system
  - (e.g., Windows, Mac OS X, Linux has many, such as Gnome, XFCE, …)
- The WIMP paradigm: Windows, Icons, Menus, Pointers
- If an app wishes to use graphics, then the app requires a container (window) for the graphics

How the WM works (in a nutshell)

- The app **requests** a window from the window manager
- The window manager **decides** whether a window is shown
  - It is not up to the app, the app **cedes** autonomy to the WM
  - The WM allows the user to minimize, overlap, maximize the windows on the desktop
- The app can **ask** the WM about its screen real estate (which can change over time)
- The app **tells** the WM that it is in need of redrawing (`repaint`);
  - the WM decides to redraw all of some of its windows

Separation of Concerns

- The app specifies what should be drawn (the **WHAT**)
- The WM actually does the drawing (the **HOW**)
- As the app developer, you need to understand this separation



YORK UNIVERSITÉ UNIVERSITY

`Graphics2D` class services

- the `Graphics2D` object encapsulates the "HOW" part of the drawing
- the complexity of the "HOW" is hidden from the clients
  - all of the low level stuff that concerns graphics rendering is hidden, e.g., how to translate drawing coordinates to screen coordinates, how to set the sub-pixel values in order to accomplish the different colours, etc

YORK UNIVERSITÉ UNIVERSITY

`Graphics2D` services

- an app that **has** a window ➔ can obtain access to the window's `Graphics2D` object
- an app that **does not** have a window ➔ **no** access to a `Graphics2D` object

- The `Graphics2D` class is part of java.awt (Abstract Window Toolkit)
- This `Graphics2D` class extends the `Graphics` class to provide more sophisticated control over geometry, coordinate transformations, color management, and text layout.

Obtaining the `Graphics2D` reference

Suppose we have a `Picture` object with reference `myPict`

Obtain a reference to window's `Graphics2D` object:

```
Graphics2D graphicsObj = myPict.createGraphics();
```

How do we use the graphics2D object?

Examples

Suppose we have a `Picture` object with reference `myPict`

Obtain a reference to window's Graphics2D object:

```
Graphics2D graphicsObj = myPict.getGraphics();
```

Now we specify to the graphics2D object what primitives we want drawn

`Shape` is part of java.awt as well.

```
Shape shape1;
// instantiate shape1…
graphicsObj.draw(shape1);
graphicsObj.fill(shape1);
```

Instantiating a Shape object

```
int width = 20;
int height = 50;
int xPos = 5;
int yPos = 15;
Rectangle2D.Double shape1 =
                    new Rectangle2D.Double(xPos, yPos, width, height);
```

The Rectangle shape is 20 units wide and 50 units high.

The upper left hand corner is anchored at (5,15)

Instantiating a `Shape` object

```
Rectangle2D.Double shape1 =
                    new Rectangle2D.Double(xPos, yPos, width, height);
```

The `Rectangle2D.Double` shape is 20 units wide and 50 units high.

What are these units?

The units are “coordinate units” in the **user space**.

User Space

- *User space* is the coordinate space in which graphics primitives are specified
  - a device-independent logical coordinate system.
  - the coordinate space that your program uses.
  - The origin of user space is the upper-left corner of the component's drawing area.
- All geometries passed into Java 2D rendering routines are specified in **user-space coordinates**.
- When it is time to render the graphics, a transformation is applied to convert from user space to **device space**.

http://docs.oracle.com/javase/tutorial/2d/overview/coordinate.html

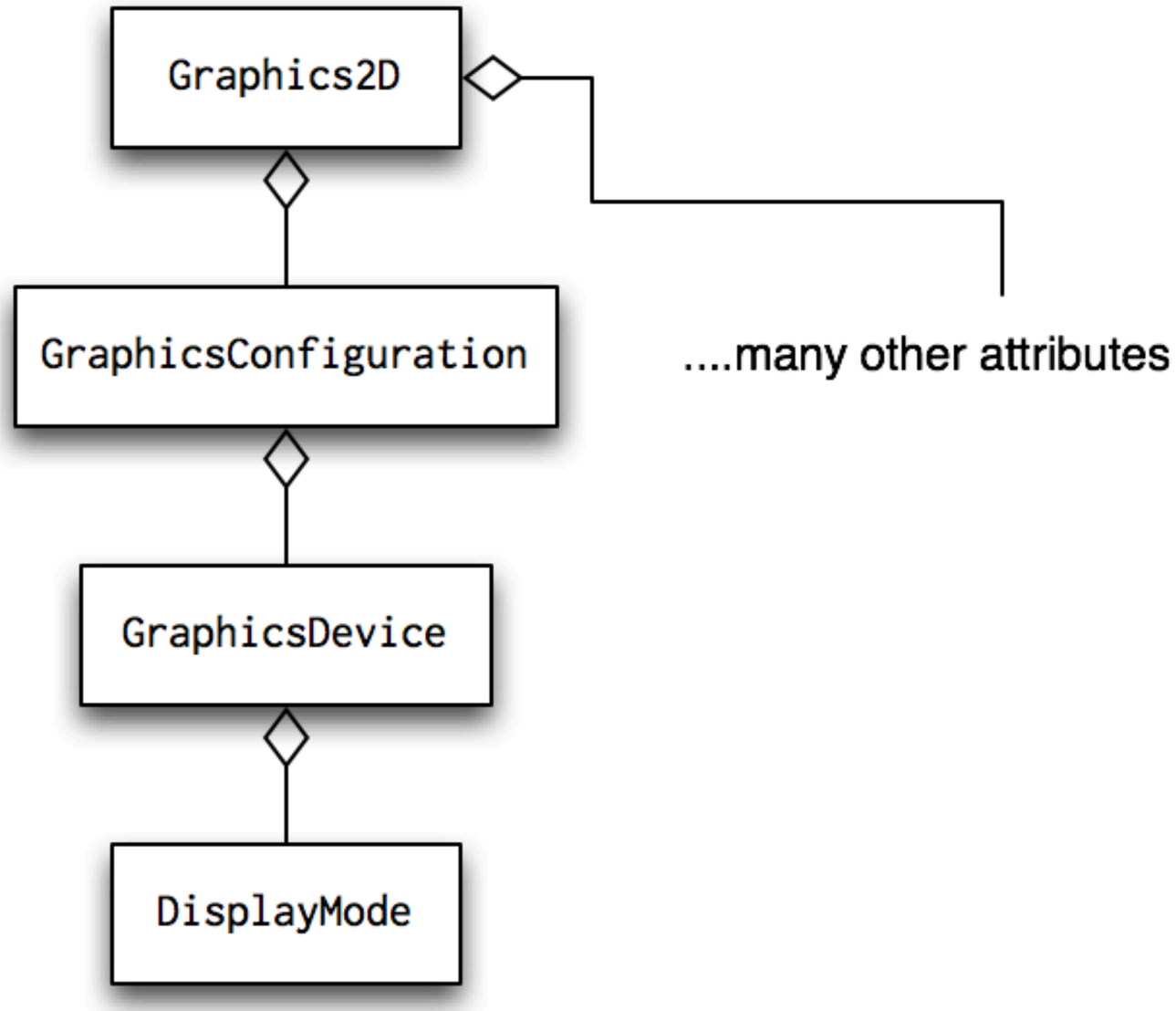
Device Space

- *Device space* – The coordinate system of an output device such as a screen, window, or a printer
- Your app can invoke the following to determine the screen resolution in dots per inch:
  `Toolkit.getDefaultToolkit().getScreenResolution()`
- Depending on the screen resolution, one point in user space may translate to several pixels in device space
- If your screen resolution is 72, then there is likely to be 72 “coordinate units” in user space per inch. But this can vary.

http://docs.oracle.com/javase/tutorial/2d/overview/coordinate.html

User Space

- The client specifies the graphic primitives to be drawn in *user space* (in “coordinate units”)
- `Graphics2D` class services translates the coordinates in *user space* to coordinates in *device space* (in pixels)



see L07_Ex2

....current DisplayMode has width, height

YORK UNIVERSITÉ UNIVERSITY

Instantiating a `Shape` object

```
Rectangle2D.Double shape1 =
                    new Rectangle2D.Double(xPos, yPos, width, height);
```

The name of the class is weird – there is a dot in the middle of it.

`Rectangle2D.Double` is a subclass of the class `Rectangle2D`

`Rectangle2D` is a subclass of the class `Shape`

Neither `Shape` nor `Rectangle2D` have constructors

“When a parent is expected, a child is accepted” (Ch 9)

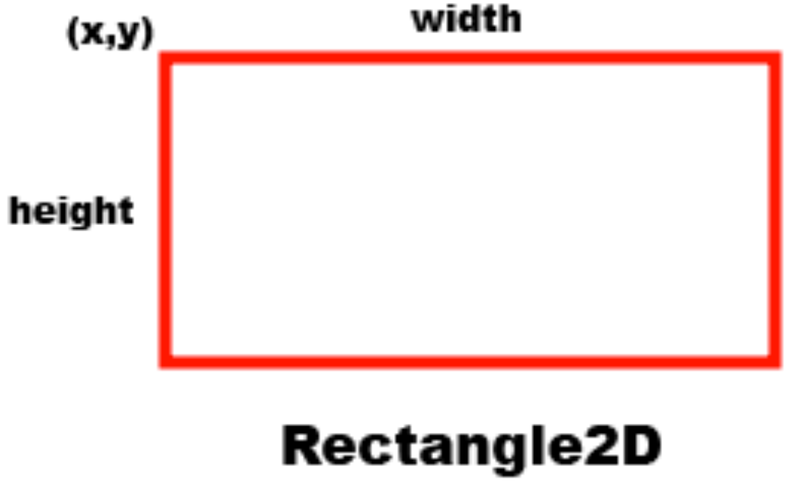
This is the “substitutability principle”

```
Rectangle2D.Double shape1 =
                new Rectangle2D.Double(xPos, yPos, width, height);
Rectangle2D shape1 =
                  new Rectangle2D.Double(xPos, yPos, width, height);
Shape shape1 =
                  new Rectangle2D.Double(xPos, yPos, width, height);
```

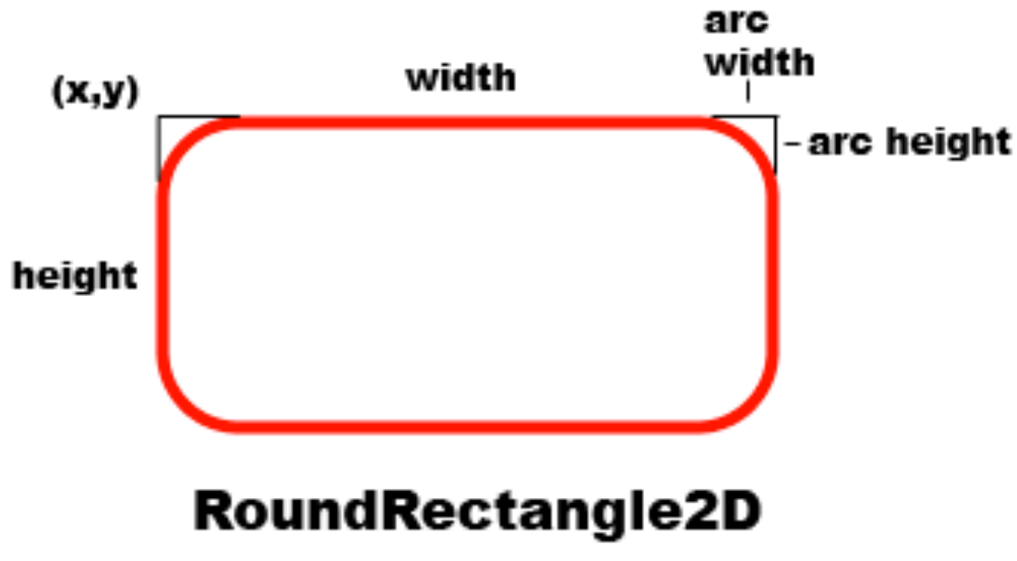
`Graphics2D` primitives

- basic geometric shapes: `draw(Shape)` `fill(Shape)`
- lines: `drawLine(int, int, int, int)`
- text: `drawString(String, int, int)`

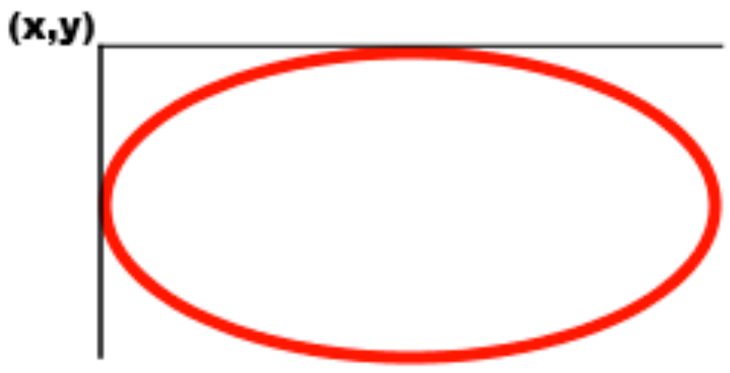
Shape Primitives



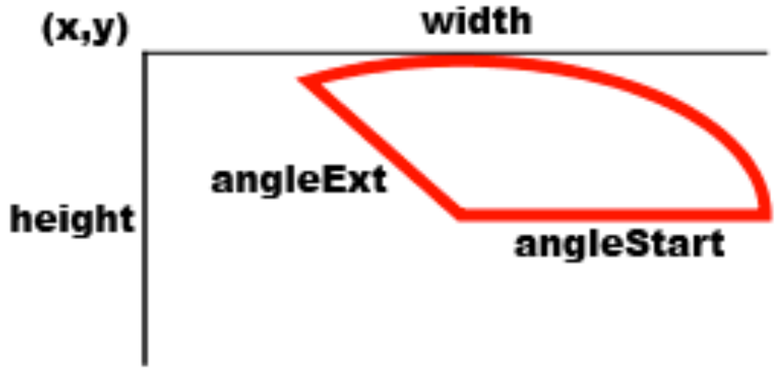
Rectangle2D



RoundRectangle2D



Ellipse2D

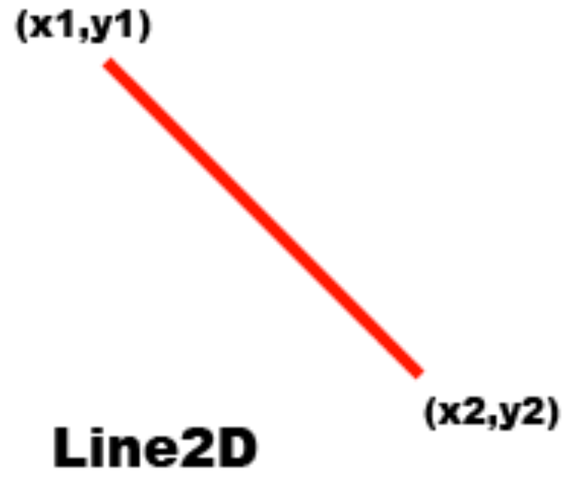


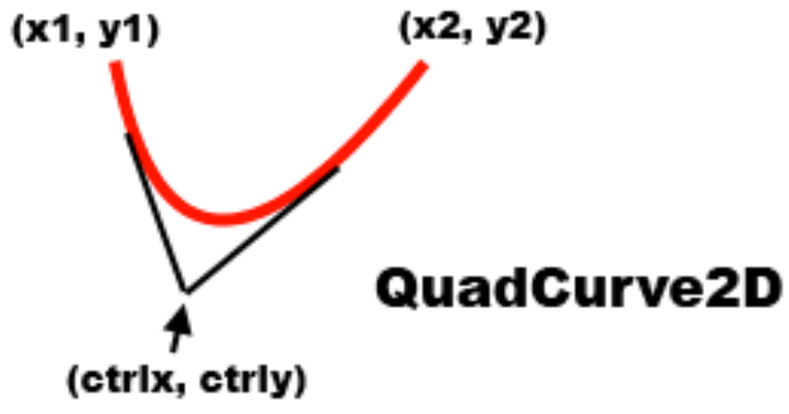
Arc2D

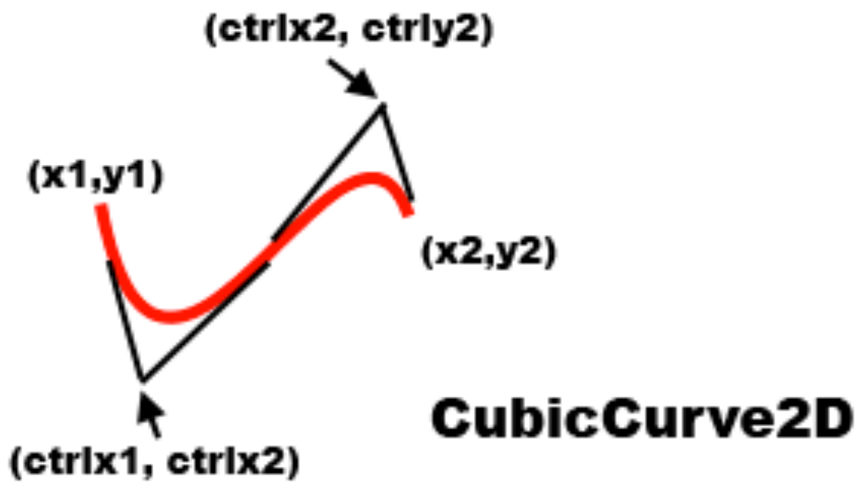
21

Shape Primitives

ref: http://java.sun.com/developer/technicalArticles/GUI/java2d/java2dpart1.html





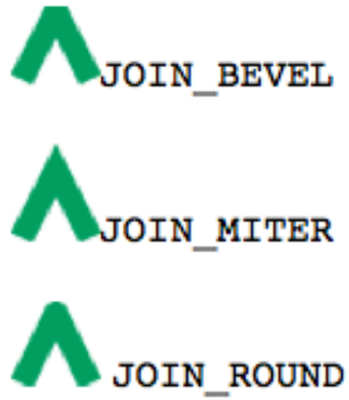


22

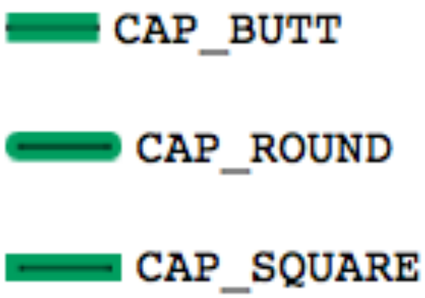
Current Settings (II)

Additional settings:

- the way the strokes are joined together



- the appearance of the ends of lines



- The current translation, rotation, scaling, and shearing values

Current Settings (I)

- when any primitive is drawn, it is drawn with the **current settings** of the `Graphics2D` object.
- any primitive that is drawn is *drawn with the current settings* until the settings change
- the settings are determined by attribute values
  → thus we say that the **state** of the `Graphics2D` object determines the drawing settings.

The settings include:

- the `Paint` to be used (the colour of the drawing pen)
- the `Stroke` to be used (the width of the drawing pen)
- the `Fill` to be used (the pattern used to fill the shapes)

Example: changing pen colour

```
graphicsObj.setPaint(Color.BLUE);
graphicsObj.draw(shape1);
graphicsObj.setPaint(Color.RED);
graphicsObj.draw(shape1);
```

This draws a red rectangle on top of the blue rectangle

Any shape that is drawn is drawn with the current settings until the settings change

About Colour

- Paint controls the colour of the drawing pen
- The default colour is WHITE
- Here's how to change it (newer, better version):

```
graphicsObj.setPaint(Color.BLUE);
```

- An older version:

```
graphicsObj.setColor(Color.BLUE);
```

the `setPaint` method takes a `Paint` argument, and a `Color` object can fit the bill!

Here is a fancier fill

```
Point p1 = new Point(0, 0);
Point p2 = new Point(50, 50);
GradientPaint paint1 =
       new GradientPaint(p1, Color.RED, p2, Color.MAGENTA,
true);
graphicsObj.setPaint(paint1);
```

Try it yourself!

Example: changing pen width

- Stroke controls the width of the drawing pen
- The default width is 1 unit (typically 1 pixel wide, so it is teeny-tiny)
- Here's how to change it:

```
BasicStroke newStroke = new BasicStroke(4.0);
graphicsObj.setStroke(newStroke);
```

Since `Stroke` is the parent class of `BasicStroke`, you can also write:

```
Stroke newStroke = new BasicStroke(4.0);
graphicsObj.setStroke(newStroke);
```

Can we move a Shape?

Once we instantiate a shape, there is no way to “move” it.

Instead, just instantiate new shapes with different anchor points

- You can move the origin of the coordinate system up/ down or left/right
  - this will make it appear as though the anchor of the rectangle has moved
  - this is not recommended at this point, since we want a fixed origin

About transformations

Once a shape is specified in user space, then any number of **transformations** can be applied to it

For instance, here is a shear transformation of a rectangle

There are also transformations to rotate and scale.

To Do:

- Practise using all of these various methods and experiment on your own.
- Complete the lab exercise lab L8.2 (pp.329-332).
- Do exercise 8.20