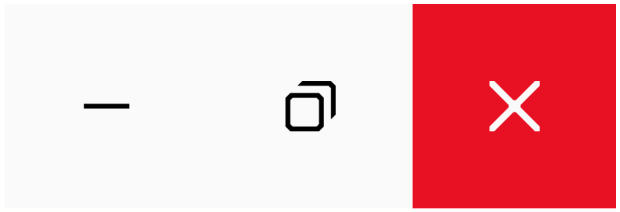
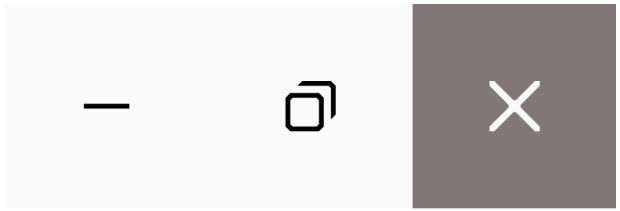


Drawing

Primitives
Graphics Context
The Painter's Algorithm

June 5



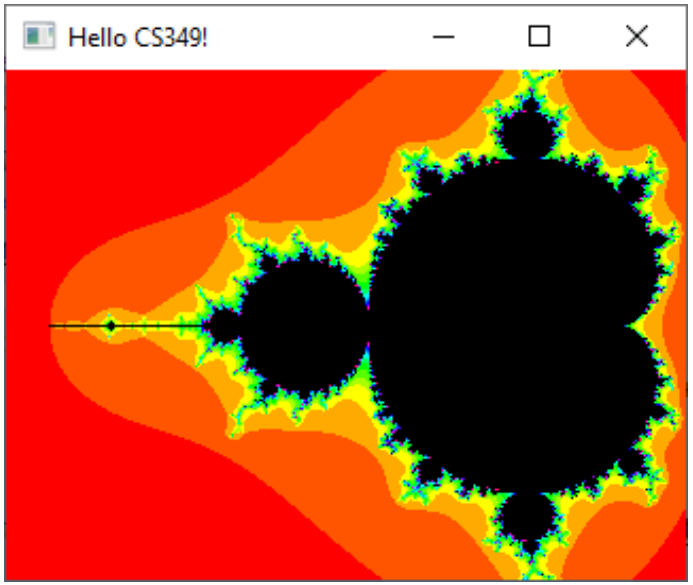
Primitives

The foundations of graphical output.

Drawing Primitives – Pixel

Three conceptual models for drawing:

1. Pixel
   - `SetPixel(x, y, color)`
   - `DrawImage(x, y, image_source)`



Drawing Primitives – Strokes

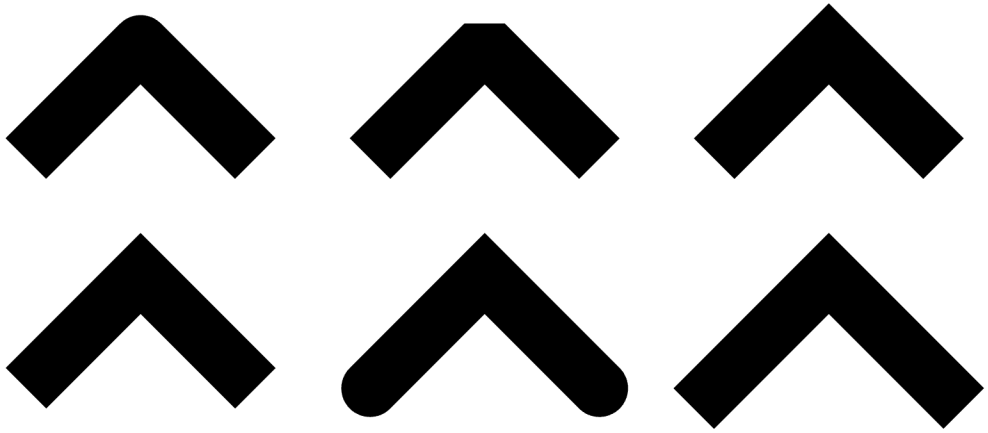
Three conceptual models for drawing:

2. Stroke
   - `DrawLine(x1, y1, x2, y2, line_style)`
   - `DrawRect(x, y, width, height, border_style)`
   - `DrawPolyline(x1, y1, x2, y2, ..., xn, yn, border_style)`
   - `DrawArc(x, y, width, height, start, end, border_style)`

Drawing Primitives – Stroke Styles

Many options:

- Colour
- Thickness
- Style: solid, dashed
- Joints: round, bevel, miter
- Cap: butt, round square



Drawing Primitives – Regions

Three conceptual models for drawing:

3. Region
   - `DrawText("CS349", x, y, text_style)`
   - `DrawRect(x, y, width, height, border_style, fill_style)`

Drawing Primitives – Region Styles

Some fill options:

- Colour: solid, gradient, pattern
- Opacity

Some text options:

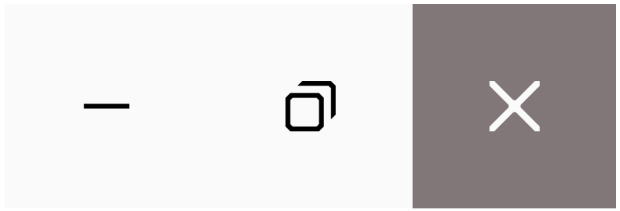
- Family or Name
- Size
- Weight: thin, …, black
- Slope: normal, italic

abcde

JavaFX Primitives

Primitives drawing commands include:

- **Rectangles**: `strokeRect()`, `fillRect()`
- **Rounded rectangles**: `strokeRoundRect()`, `fillRoundRect()`
- **Oval (and Circle)**: `strokeOval()`, `fillOval()`
- **Polygons**: `strokePolygon()`, `fillPolygon()`
- **Arcs**: `strokeArc()`, `fillArc()`
- **Text**: `strokeText()`, `fillText()`
- **Line**: `strokeLine()`, `strokePolyline()`



Graphics Context

Drawing using a Graphics Context (GC)

JavaFX Canvas and Graphics Context

The JavaFX Canvas Node provides a drawing surface (aka canvas).

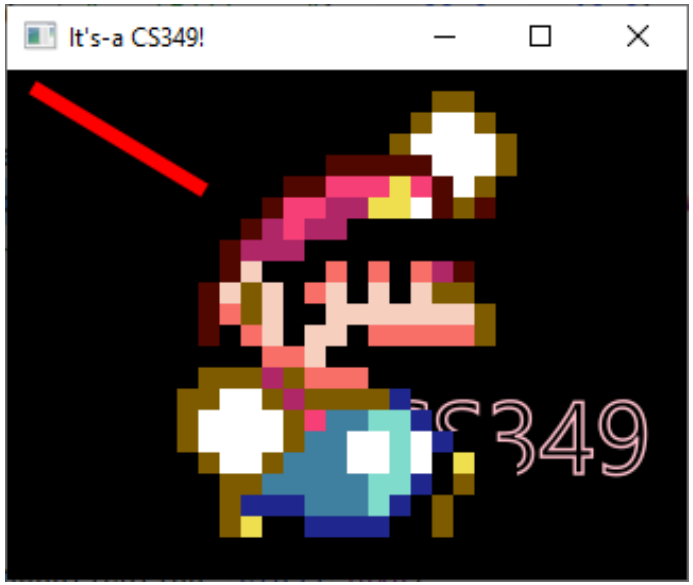
- it has a buffer where the drawing is rendered
- It has a single Graphics Context that is modified to set drawing parameters state (e.g., `border_style`) and issue drawing commands to.

```
val canvas = Canvas(320.0, 240.0)
canvas.graphicsContext2D.apply {

    stroke = Color.RED
    lineWidth = 7.0
    strokeLine(15.0, 10.0, 90.0, 55.0)

    stroke = Color.PINK
    lineWidth = 1.5
    font = Font.font("Console", 48.0)
    strokeText("CS349", 170.0, 190.0, 180.0)

    drawImage(Image("plumber.png"), 80.0, 10.0)
}
```

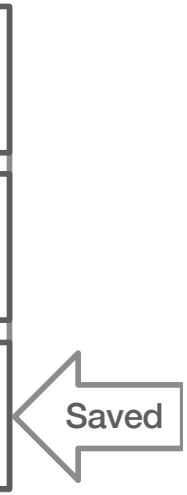


JavaFX Canvas and Graphics Context

A graphics context contains drawing parameters and all device-specific information that the drawing system needs to perform any subsequent drawing commands.

The graphics content can be pictured as a stack, and it is possible to save and restore previous states.

```
stroke = Color.RED
lineWidth = 4.0
setLineDashes(7.0, 15.0)
save()
strokeRect(10.0, 10.0, 100.0, 10.0)
stroke = Color.GREEN
lineWidth = 2.0
setLineDashes(0.0)
strokeRect(10.0, 30.0, 100.0, 10.0)
stroke = Color.ORANGE
lineWidth = 8.0
strokeRect(10.0, 50.0, 100.0, 10.0)
restore()
strokeRect(10.0, 70.0, 100.0, 10.0)
```



JavaFX Canvas and Graphics Context

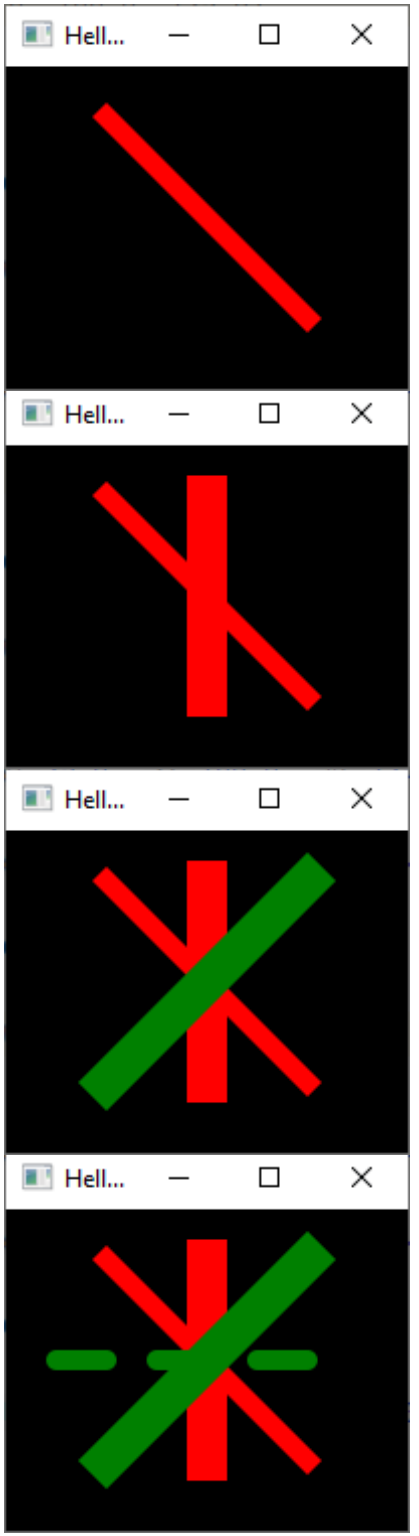
A common approach is to maintain the current state of all drawing options in a *Graphics Context*. A drawing command, e.g., `strokeLine`, is then rendered using the current set of options.

```
stroke = Color.RED
lineWidth = 10.0
strokeLine(50.0, 25.0, 150.0, 125.0)

lineWidth = 20.0
strokeLine(100.0, 25.0, 100.0, 125.0)

stroke = Color.GREEN
strokeLine(150.0, 25.0, 50.0, 125.0)

lineWidth = 10.0
lineCap = StrokeLineCap.ROUND
setLineDashes(25.0)
strokeLine(25.0, 75.0, 175.0, 75.0)
```



JavaFX Canvas and Graphics Context

Graphics context attributes include:

- Fill options
- Stroke options
- Text option
- Rendering: clipping, blend, transforms

Drawing using Graphics Context

1. Create `Canvas`
2. Use the *`graphicsContext2D`* to set drawing attributes
3. Use *`graphicsContext2D`* to draw shapes


```
override fun start(stage: Stage) {
    val canvas = Canvas(320.0, 240.0)
    canvas.graphicsContext2D.apply {
        stroke = Color.WHITE
        fill = Color.WHITE
        fillRect(75.0, 25.0, 100.0, 100.0)
        fill = Color.AQUA
        fillRect(100.0, 50.0, 100.0, 100.0)
        fill = Color.MEDIUMPURPLE
        fillRect(125.0, 75.0, 100.0, 100.0)
    }
    stage.apply {
        scene = Scene(Group(canvas), 320.0, 240.0, Color.BLACK)
        title = "Hello CS349!"
    }.show()
}
```

Drawing using Convenience Classes

JavaFX Nodes *also* include Shape classes, which Can be drawn directly, i.e., without a Canvas.

```
override fun start(stage: Stage) {
    val rectList = listOf(
        Rectangle(100.0, 100.0, Color.WHITE).apply {
            x = 75.0; y = 75.0 },
        Rectangle(100.0, 100.0, Color.AQUA).apply {
            x = 100.0; y = 100.0 },
        Rectangle(100.0, 100.0, Color.MEDIUMPURPLE).apply {
            x = 125.0; y = 125.0 }
    )
    stage.apply {
        scene = Scene(Group(rectList), 320.0, 240.0, Color.BLACK)
        title = "Hello CS349!"
    }.show()
}
```

Why would you use these over a canvas? We'll discuss next lecture!

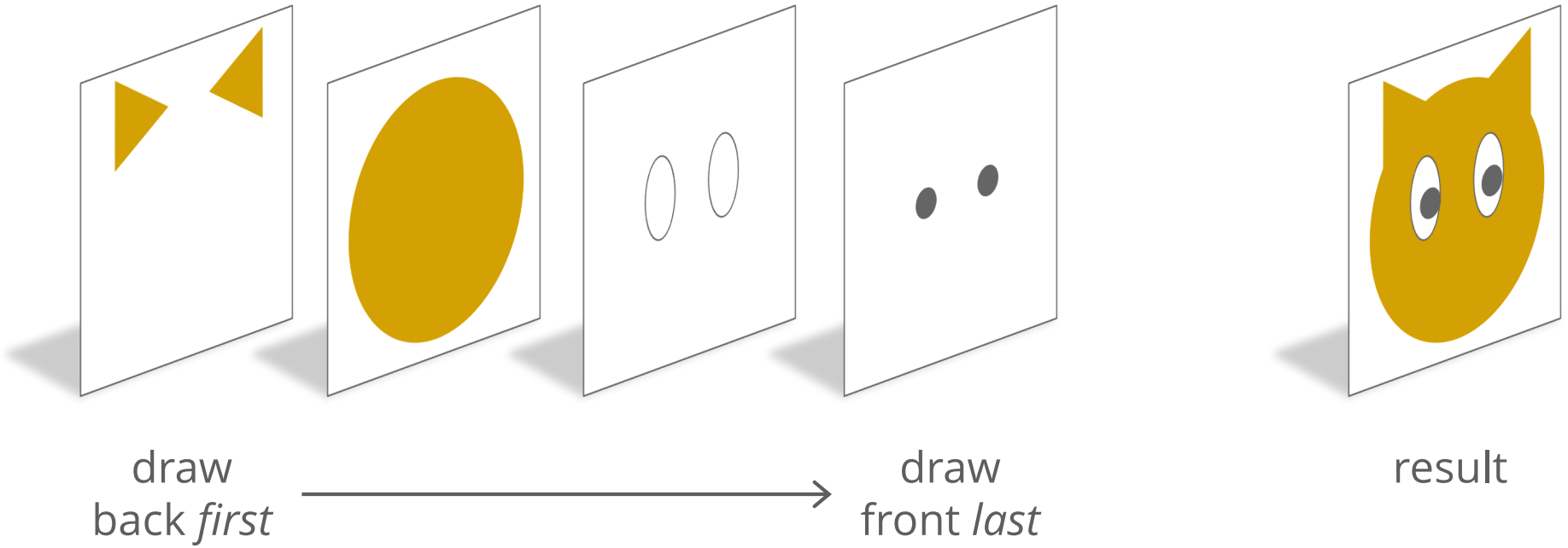
The Painter’s Algorithm

Painter's Algorithm

Basic graphics primitives are (really) *primitive*. To draw more complex shapes:

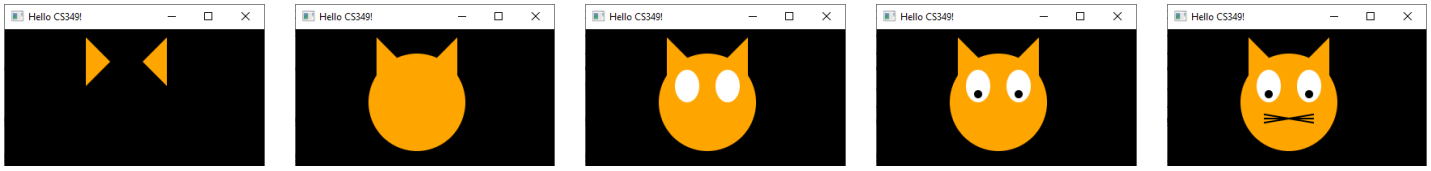
- Combine primitives
- Draw back-to-front, layering the image

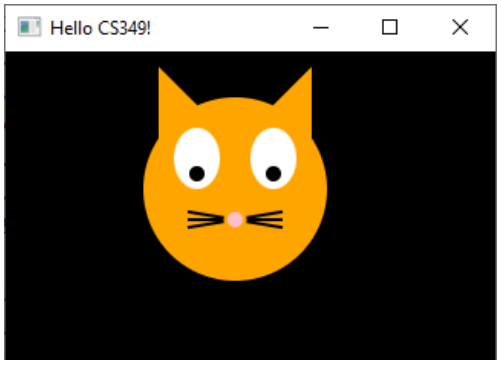
This approach is called "Painter's Algorithm".

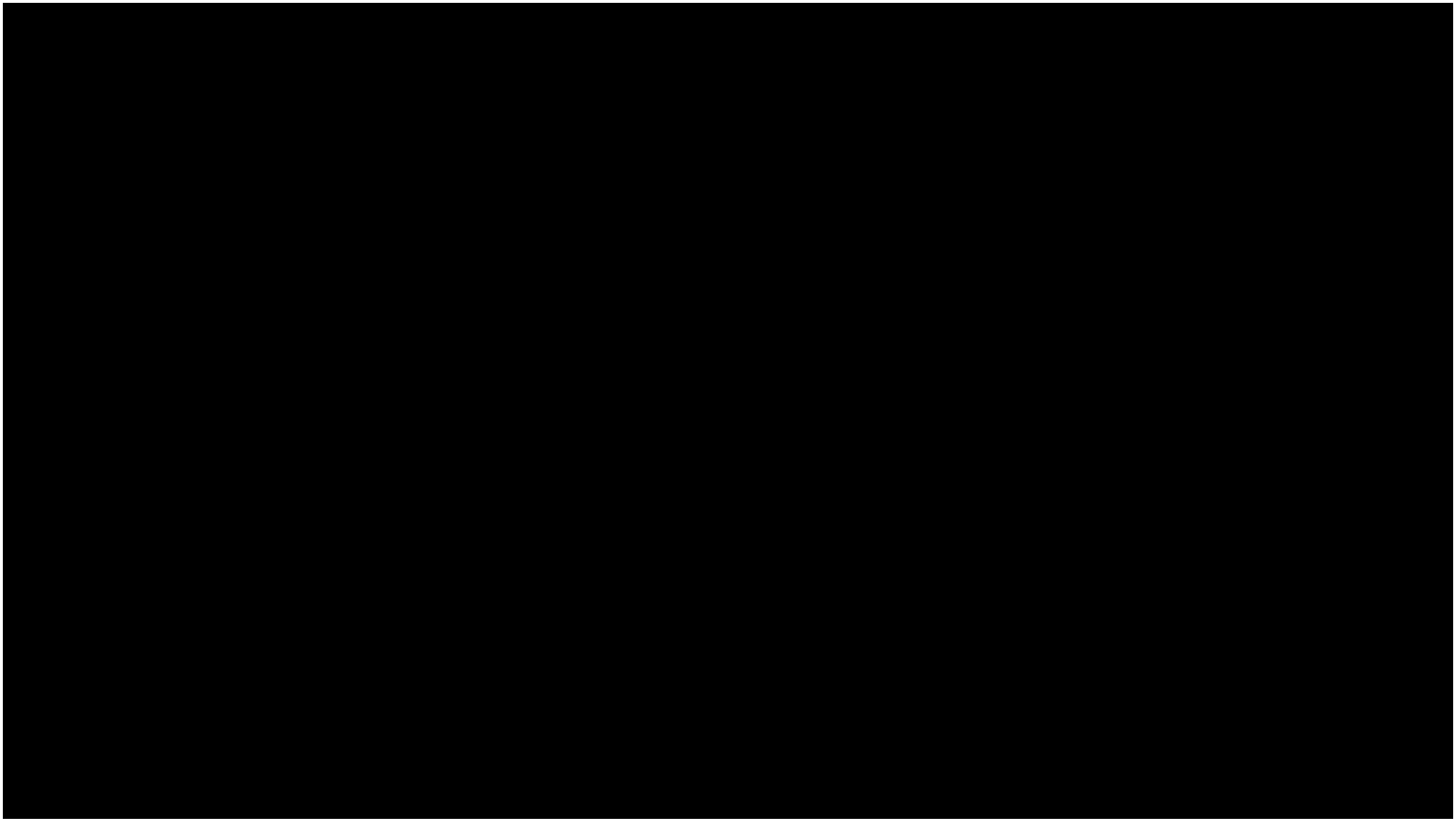


Painter's Algorithm

```
canvas.graphicsContext2D.apply {
    fill = Color.ORANGE
    fillPolygon(mutableListOf(100.0, 130.0, 100.0).toDoubleArray(),
        mutableListOf(10.0, 40.0, 70.0).toDoubleArray(), 3)
    fillPolygon(mutableListOf(200.0, 170.0, 200.0).toDoubleArray(),
        mutableListOf(10.0, 40.0, 70.0).toDoubleArray(), 3)
    fillOval(90.0, 30.0, 120.0, 120.0)
    fill = Color.WHITE
    fillOval(110.0, 50.0, 30.0, 40.0)
    fillOval(160.0, 50.0, 30.0, 40.0)
    fill = Color.BLACK
    fillOval(120.0, 75.0, 10.0, 10.0)
    fillOval(170.0, 75.0, 10.0, 10.0)
    stroke = Color.BLACK
    lineWidth = 2.0
    strokeLine(120.0, 105.0, 180.0, 115.0)
    strokeLine(120.0, 110.0, 180.0, 110.0)
    strokeLine(180.0, 105.0, 120.0, 115.0)
    fill = Color.PINK
    fillOval(145.0, 105.0, 10.0, 10.0)
}
```







Amazing City spray paint art
https://www.youtube.com/watch?v=k4gH7xx3tVc

Drawable Objects & Their Interface

We refer to an object's "depth" in the stack as it's "z-index"

- x, y are 2D positional coordinates, so z is depth

To create a shape that can stack under/over other shapes, extend a shape class with `z-index` information

```
class RectangleZ(x: Double, y: Double,
                 width: Double, height: Double,
                 fill: Paint, val zindex: Int,
) : Rectangle(x, y, width, height) {
    init {
        this.fill = fill
        stroke = Color.WHITE
    }
}
```

List of Displayables

```
val rectList = listOf(
    RectangleZ(75.0, 25.0, 100.0, 100.0, Color.WHITE, 1),
    RectangleZ(125.0, 75.0, 100.0, 100.0, Color.MEDIUMPURPLE, 3)
    RectangleZ(100.0, 50.0, 100.0, 100.0, Color.AQUA, 2),
)

val root = Group(rectList.sortedBy {  // sort by depth
    rectangleZ -> rectangleZ.zindex
})
```

End of the Chapter

- How to draw primitives using the `Canvas`
- How to draw primitives using Convenience Classes