Java 2D: Introduction

What is Java 2D

Set of classes that can be used to create graphics;
Features: geometric transformation, antialiasing, alpha compositing, image processing, text manipulation, …
Part of the core classes since Java 2.

What is Java 2D

The API encompasses the following packages:

- java.awt,
- java.awt.image,
- java.awt.color,
- java.awt.font,
- java.awt.geom,
- java.awt.print

What can be done?

Shapes: toolbox of standard shapes, arcs, paths, curves
Stroking: lines can be drawn as solid or dotted line with different thickness
Filling: closed shapes can be filled with color (solid, gradient, pattern)
Transformation: toolbox of usual transformations; can be applied on anything (shape, text, image)

What can be done?

Text: strings can be rendered and manipulated the same ways shapes are;
Antialiasing: for reducing jagged edges;
Images: load, inspect, manipulate images with transformation;
Image processing: classes for implementing basic filters;
Alpha compositing, color manipulation, clipping.

What can be done?

Graphics and Graphics2D

Everything (shape, text, image) is drawn on a Graphics2D object (Graphics for older Java version);
Any Component (AWT) or JComponent (Swing) has a paint() method that receives a Graphics object (actually a Graphics2D in version > 2);
Drawing is done in the overriden paint() method

Drawing on Component

```
@Override
public void paint(Graphics g) {
 Graphics2D g2 = (Graphics2D)g;
 // draw on the component by calling
 // methods on g
}
```

Drawing on Component (alternative)

```
@Override
public void paintComponent(Graphics g) {
 Graphics2D g2 = (Graphics2D)g;
 // draw on the component by calling
 // methods on g
}
```

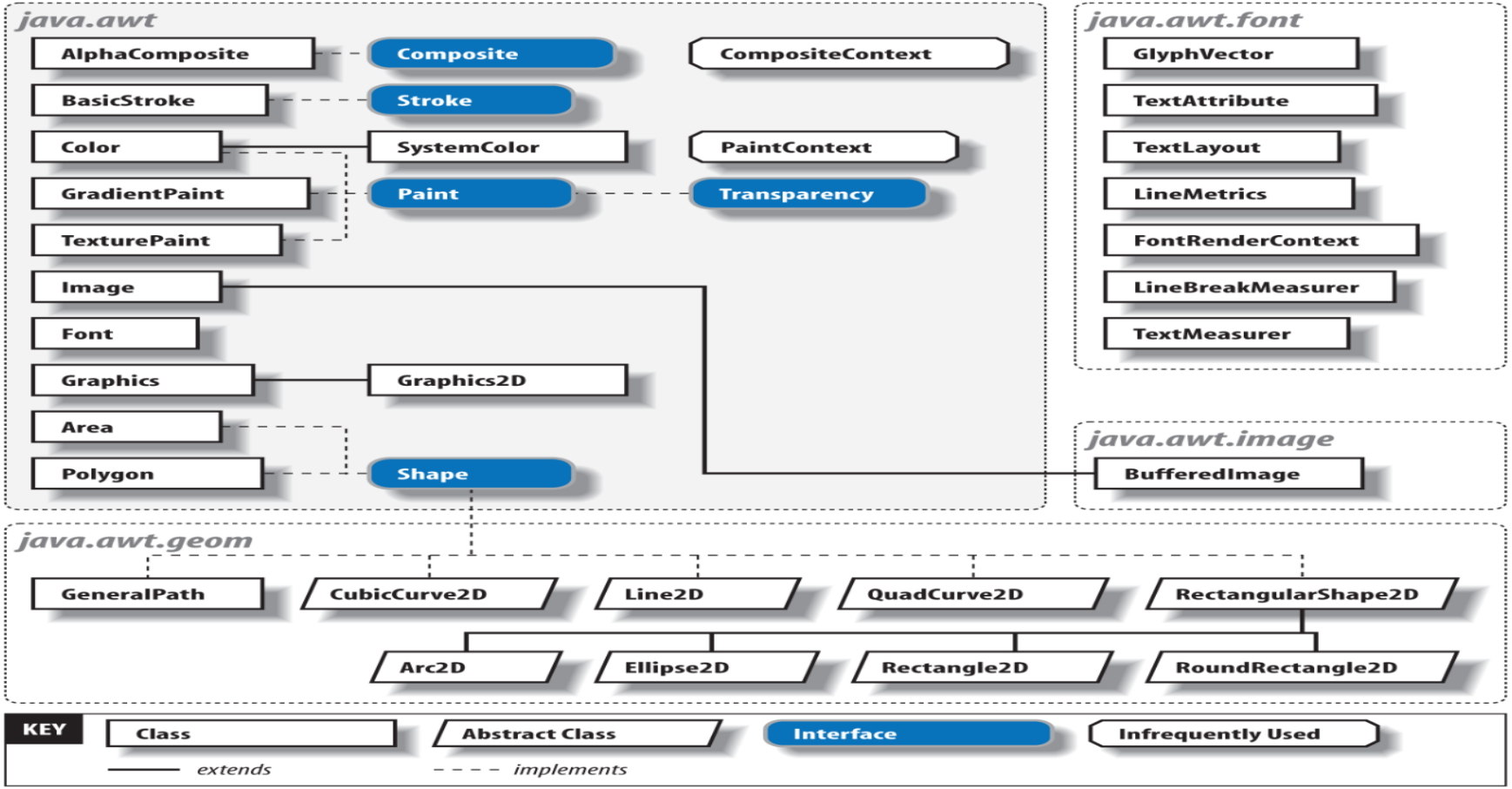
Drawing on Image

Similarly one can draw on images with:

```
public void drawOnImage(Image i) {
  Graphics g = i.getGraphics();
  // ...
}
```

or with BufferedImage (preferred):

```
public void drawOnImage(BufferedImage i) {
  Graphics2D g2 = i.createGraphics()
  // ...
}
```


java.awt
AlphaComposite
Composite
CompositeContext
BasicStroke
Stroke
Color
SystemColor
PaintContext
GradientPaint
Paint
Transparency
TexturePaint
Image
Font
Graphics
Graphics2D
Area
Polygon
Shape
java.awt.font
GlyphVector
TextAttribute
TextLayout
LineMetrics
FontRenderContext
LineBreakMeasurer
TextMeasurer
java.awt.image
BufferedImage
java.awt.geom
GeneralPath
CubicCurve2D
Line2D
QuadCurve2D
RectangularShape2D
Arc2D
Ellipse2D
Rectangle2D
RoundRectangle2D
KEY
Class
Abstract Class
Interface
Infrequently Used
extends
implements

Graphics2D

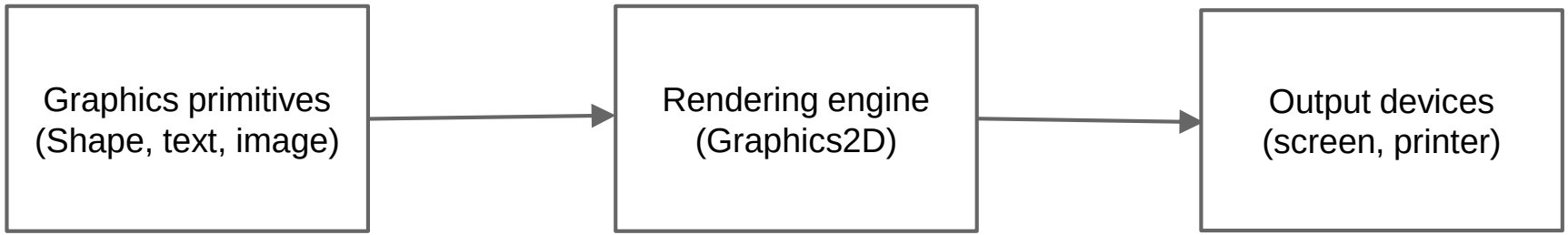
Rendering is the process of taking shapes, text, … and drawing the corresponding pixels on a screen;
Shape, text, image are graphics primitives;
Screens, printers are output devices;
A rendering engine performs the rendering; in the 2D API Graphics2D class implements such a rendering engine.

Graphics2D

Four ways to obtain a Graphics2D object:

- From AWT or Swing as a result of a painting request;
- From an offscreen image buffer;
- From copying an existing Graphics2D object;
- From an onscreen component.

The rendering pipeline



The rendering pipeline

Conversion from graphics primitives to pixel color is decided by the internal state of the Graphics2D object;
This state is defined by:
paint: determines what colors to use to fill a shape;
stroke: determines how the outline of the shape is drawn;
font: determines what shapes are created for a given set of characters;

The rendering pipeline

This state is defined by:
transformation: determines how to transform the primitives before they are rendered; converts primitives from user space to device space;
compositing rule: how the colors of a primitive are combined with existing color in the color buffer;
clipping shape: rendering operations are limited to the interior of the clipping shape;
rendering hints: hints on rendering technique to use.

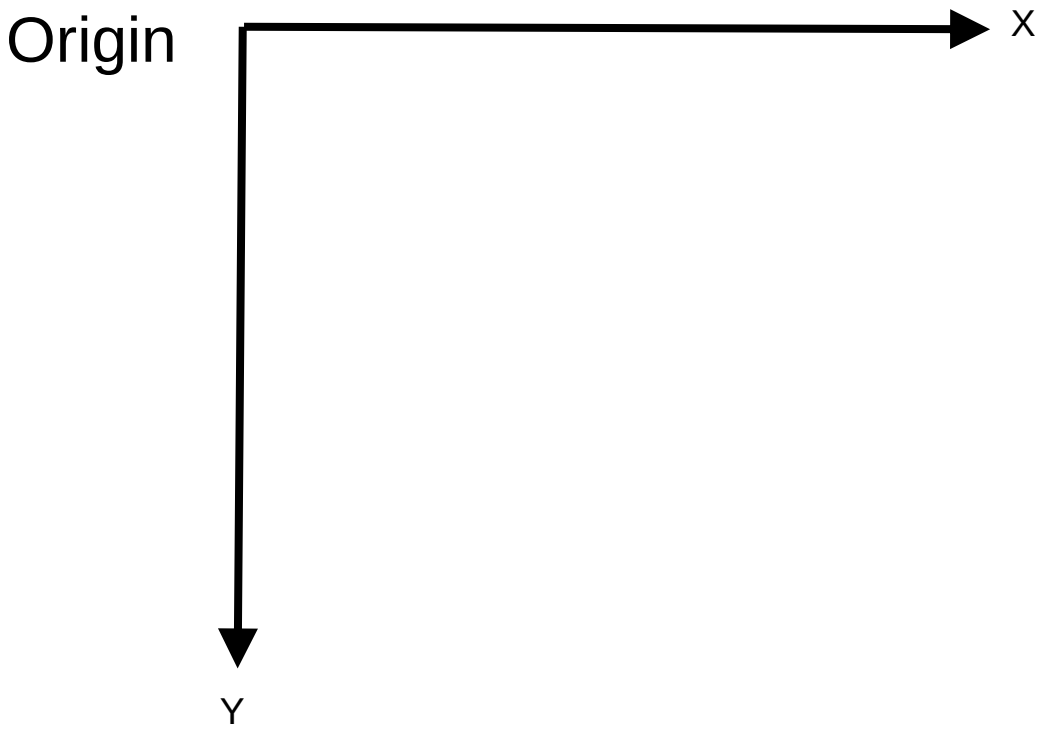
The rendering pipeline

1. Determine the shape to be rendered; apply the given transformation to it;
2. Rasterize the shape; rendering hints are used to control the behavior of the rasterization;
3. Clip the result with the clipping shape;
4. Determine the colors to use;
5. Combine the color of the incoming primitive with the existing color using the compositing rule.

Coordinate space

Java 2D objects live in a plane defined by Cartesian coordinates (**User Space**);
When drawn to screen, User Space coordinates are transformed in **Device Space** Coordinates;
One unit in Device Space corresponds to one pixel on this device; Device Space is dependent on a particular rendering device;

Coordinate system



Device Space

By default, Device Space and User Space are aligned;
Scaling is applied to ensure that objects are drawn with the same size regardless of the output device;
User space is converted to Device Space when objects are drawn; a transformation is used;
This is all done transparently, no need to worry about it.