COMP112/18 - Programming I

12 Graphics

Instructor: Ke Wei（柯韋）

A319 ✆ Ext. 6452 ✉ wke@ipm.edu.mo

http://brouwer.ipm.edu.mo/COMP112/18/

Bachelor of Science in Computing, School of Public Administration, Macao Polytechnic Institute

October 12, 2018

Outline

Package Kon for Graphics and Console

About Kon

- The Kon package provides simplified 2D graphics functions for students to get started.
- Kon displays a UI window with a *console* section and a *canvas* section.
- Students can write programs to draw graphics on the canvas while performing console I/O. In addition, the text in the console and the graphics shown on the canvas can be saved to files through UI commands.
- The Kon package (`mo.edu.ipm.esap.kon`) can be downloaded as a ***JAR*** archive together with the ***source files*** and the ***javadoc*** files. The javadoc files can also be browsed ***online***.
- The Kon package is licensed under ***GPL3***.

Installation and Configuration of Kon

Installation

- Put the downloaded JAR files into a local folder on your computer.

Project configuration

- Configure the **Build Path** of your project to add the package JAR file as an **External JAR** file.
- Expand the Kon JAR file item in the **Libraries**.
- Edit the **External location** of the **Source attachment** sub-item to point to the Kon source JAR file.
- Edit the **Archive path** of the external **Javadoc location** sub-item to point to the Kon javadoc JAR file, and set the **Path within archive** to "`doc`".

Using Kon

Importing the package

- `import` *mo.edu.ipm.esap.kon.*`;`

Connecting to and using Kon in a program

- Invoke the *connect* method of class *Kon* to **initialize** the UI window of Kon,

  Kon`.`*connect*`();`

- then, you can **draw** something on the canvas,

  Kon`.`*drawLine*`(0,` *Kon*`.`*getCanvasHeight*`(),` *Kon*`.`*getCanvasWidth*`(), 0);`

- and **show** your canvas in the UI window by the *refresh* method.

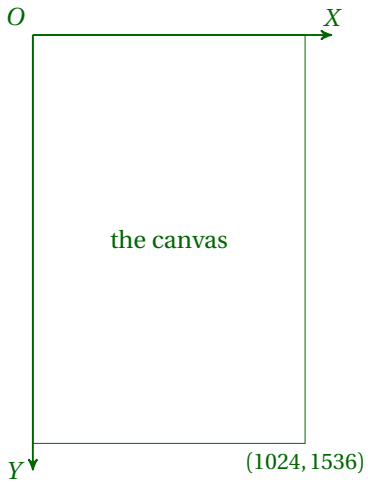
  Kon`.`*refresh*`();`

- Also, the standard input and output are redirected to the console section of Kon. For example, the following will show the text in Kon instead of the system console window.

  System`.`*out*`.`*println*`("Welcome to Kon!");`

Canvas and User Coordinate Space

- All graphics elements, including lines, curves and shapes are drawn on the rectangular canvas.
- Positions in the canvas are specified by XY coordinates.
- The upper-left corner of the canvas is the origin $(0, 0)$
- The X-coordinate increases from left to right, and the Y-coordinate increases from top to bottom.
- The width of the canvas is 1024, and the height of the canvas is 1536.
- Coordinates are of type `double`, thus can be fractional.
- Drawings on the canvas are not shown in the UI window until *Kon*`.`*refresh*`()` is invoked. You can thus draw several shapes and show them all at once.



Lines and Rectangles

- We draw a line segment by *drawLine* with two end points.

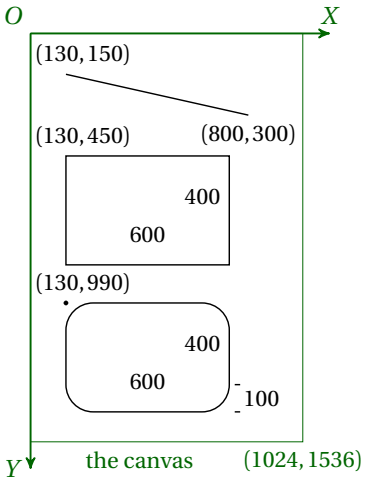
  Kon.*drawLine*(130, 150, 800, 300);

- We draw a rectangle by *drawRectangle* with the upper-left corner and the width and height.

  Kon.*drawRectangle*(130, 450, 600, 400);

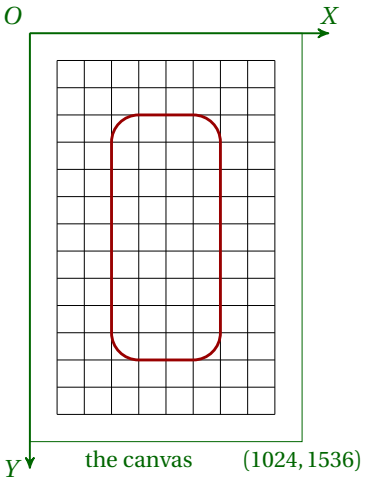
- We draw a rectangle with rounded corners by *drawRoundRectangle* with the upper-left corner, the width and height of the rectangle, and the width and height of the rounded off corners.

  Kon.*drawRoundRectangle*(130, 990, 600, 400, 100, 100);



Drawing a Grid

```
import mo.edu.ipm.esap.kon.*;
public class Grid {
  public static void main(String[] args) {
    Kon.connect();
    for ( int x = 1; x <= 9; x++ )
      Kon.drawLine(x*100, 100, x*100, 1400);
    for ( int y = 1; y <= 14; y++ )
      Kon.drawLine(100, y*100, 900, y*100);
    Kon.setColor(ColorName.DARKRED);
    Kon.setStroke(StrokeAttribute.VERYTHICK);
    Kon.drawRoundRectangle(300, 300, 400, 900,
            100, 100);
    Kon.refresh();
  }
}
```



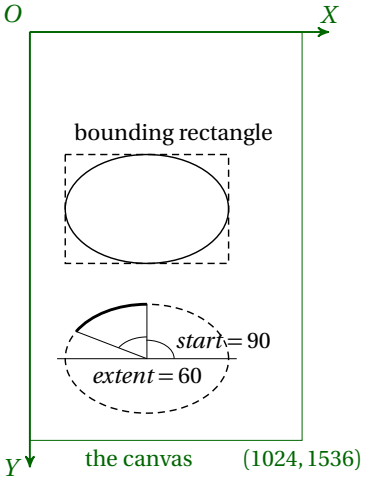
Ellipses and Arcs

- We draw an ellipse by *drawEllipse* with the upper-left corner and the width and height of the bounding rectangle

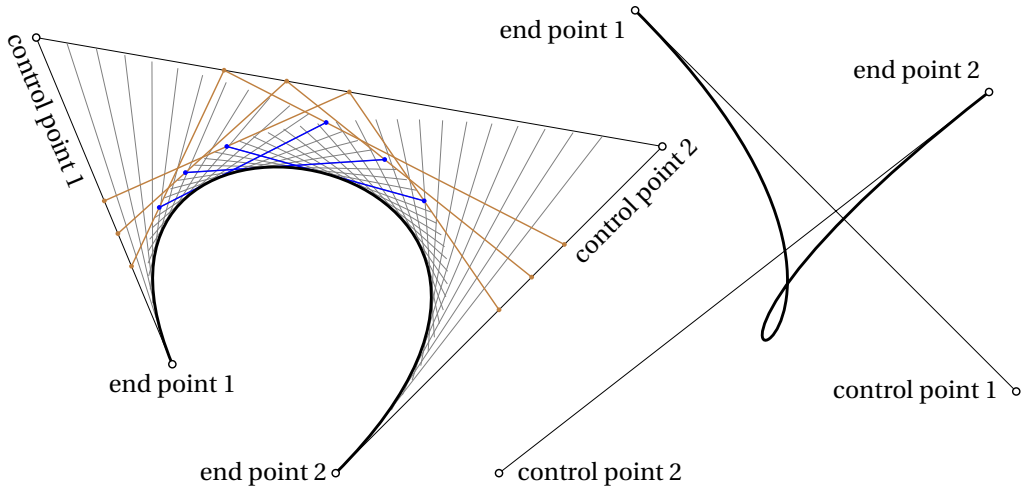
  Kon.*drawEllipse*(130, 450, 600, 400);

- We draw an arc as a part of an ellipse by *drawArc* with the upper-left corner and the width and height of the bounding rectangle of the ellipse, and the start and extent angles of the arc.

  Kon.*drawArc*(130, 1000, 600, 400, 90, 60);



Cubic Bézier Curves



Drawing Cubic Curves

- We draw a cubic curve by *drawCubicCurve* with two end points and two control points. We pass these points as follows, 1) the first end point, 2) the first control point, 3) the second control point and finally 4) the second end point.
- The curve on the left of the previous slide can be drawn as

```
Kon.drawCubicCurve(300, 800, 50, 200, 1200, 400, 600, 1000);
```

- The curve on the right of the previous slide can be drawn as

```
Kon.drawCubicCurve(250, 150, 950, 850, 0, 1000, 900, 300);
```


Absolute, Relative and Abstract Points

- Until now, we specify a point by its (x, y) coordinate relative to the origin $(0, 0)$. This is called an absolute point.
- Sometimes we'd like to specify a point by its offset (x_Δ, y_Δ) relative to a reference point (x_0, y_0). This is called a relative point.
- There are two types of relative points, (x_Δ, y_Δ) — Cartesian and angle-radius — polar.
- To unify the two cases, we introduce the concept of abstract points. An abstract point can be absolute or relative. If an absolute point is relative, its reference point is interpreted by the drawing method that accepts the points.
- For example, in the *drawCubicCurve* method, the first control point and the second end point are relative to the first end point, while the second control point is relative to the second end point.

```
Kon.drawCubicCurve( Ap.xy(300, 800),    Ap.rar(120, 600),
                    Ap.rxy(600, -600),  Ap.rxy(300, 200));
```

Setting Color and Stroke

- The drawing style is controlled by the color and stroke properties of the canvas. Once we have changed the drawing style, all upcoming drawings are affected.
- We set the color by *setColor* with a predefined *ColorName*.

```
Kon.setColor(ColorName.DARKOLIVEGREEN);
```

- Or, we can also set the color by an sRGB specification, with the specified red, green, and blue values in the range (0.0 – 1.0).

```
Kon.setColor(1.0f, 1.0f, 0.0f); // yellow
```

- A stroke style consists of two attributes, the line width and the pattern.
- We set the stroke by *setStroke* with either a predefined width or pattern or both.

```
Kon.setStroke(StrokeAttribute.VERYTHICK, StrokeAttribute.DASHED);
```

We can even specify the line width of a stroke arbitrarily by a number.

```
Kon.setStroke(10, StrokeAttribute.DENSELYDOTTED); // both attributes must be present.
```


Clearing and Saving the Canvas

- We can clear the canvas to white by *clear*.

```
Kon.clear();
```

- We can also clear the canvas to any color with a predefined *ColorName* or an sRGB color.

```
Kon.clear(ColorName.BLACK);
Kon.clear(0, 0, 1); // blue
```

- Again, the clearing won't be visible until we invoke the *refresh* method.
- The graphics displayed in the UI window (not on the canvas before a refresh) can be saved to a PDF vector file or a PNG bitmap file. We do this by the menu command

  [File/Save Graphics...]

  at any time, even during the program execution.

Drawing Text

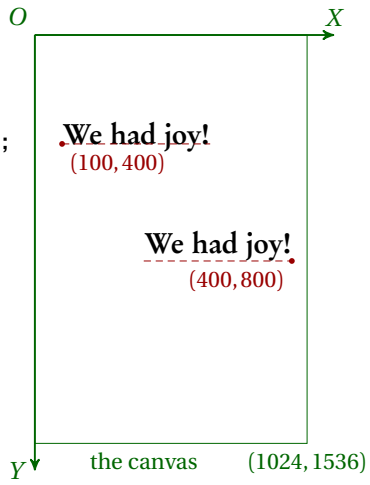
- We draw text strings using the Kon package by *drawString* with the left-most of the *baseline* at the specified position.

```
Kon.setFont("Garamond", 15, FontAttribute.BOLD);
Kon.drawString("We had joy!", 100, 400);
```

- The font can be specified by *setFont*, including the name, the size and the style.

- We can specify the anchor point by *AnchorName*.

```
Kon.drawString("We had joy!",
    AnchorName.SOUTHEAST, 400, 800);
```

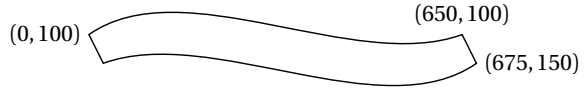


Changing Text Size and Position: Sinking Text Example

```
Sleeper sl = new Sleeper(); // An asynchronous sleeper

double cx = Kon.getCanvasWidth(), cy = Kon.getCanvasHeight();
for ( int y = 0; y <= cy; y += 4 ) {
    sl.sleep(50);     // Let the sleeper start sleeping for 50 milliseconds.
    Kon.clear();
    int fs = (int)(100*(1.1 - y / cy));
    Kon.setFont("Garamond", fs, FontAttribute.BOLD);
    Kon.drawString("We had joy!", AnchorName.NORTH, cx / 2, y);

    sl.join();     // Wait for the sleeper to wake.
    Kon.refresh();
}

sl.close(); // Close the sleeper to release the sleeper thread.
```


Constructing Paths

- Besides primitive shapes such as rectangles and ellipses, sometimes we want to define composite shapes consisting of several primitive shapes. For example, a couple of line and curve segments connected together.



- Such a composite shape is called a *path*. The above path is constructed by

```
Kon.startPath(0,100);
Kon.curveToPath(Ap.rar(35, 200), Ap.rar(-160, 200), Ap.rxy(650, 0));
Kon.lineToPath(Ap.rxy(25, 50));
Kon.curveToPath(Ap.rar(-145, 200), Ap.rar(20, 200), Ap.rxy(-650, 0));
Kon.closePath();
```

- Not only the shape, but also the position of a path is fixed. However, a path is not on the canvas until we draw it: `Kon.drawPath();`

Saving and Recovering Paths

- There is only one active path in Kon, called the current path. Starting a new path overwrites the current path.
- Kon provides four last-in, first-out stores to save and recover previously constructed paths.
- `Kon.pushPath(`i`)` pushes a copy of the current path to store (i), $0 \le i \le 3$.
- Multiple paths can be pushed onto a store, and they are popped out in reverse order.
- `Kon.popPath(`i`)` pops the topmost path from store (i) to the current path.
- The following code draws a horizontal line on the top of the canvas.

```
Kon.startPath(0,100); Kon.lineToPath(0,200); Kon.pushPath(0);
Kon.startPath(100,0); Kon.lineToPath(200,0); Kon.pushPath(0);
Kon.popPath(0); Kon.drawPath();
```

- `Kon.topPath(`i`)` copies (without removing) the topmost path of store (i) to the current path.

Changing Drawing Styles of a Path

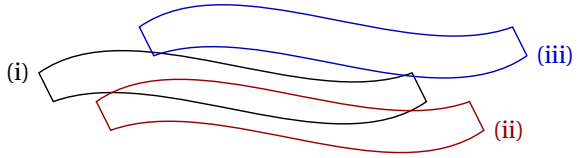
- We can change the drawing styles of a path before put it onto the canvas.
- Suppose we have constructed the path on Slide 17, this code animates the fill color.

```
1  Kon.setStroke(StrokeAttribute.VERYTHICK);
2  for ( double s = 1; s >= 0; s -= 1/128.0 ) {
3      sl.sleep(20);
4      Kon.clear();
5      Kon.setColor(s, s, s); // set sRGB color
6      Kon.fillPath();
7      Kon.setColor(ColorName.DARKRED);
8      Kon.drawPath();
9      sl.join();
10     Kon.refresh();
11 }
```


Translating Paths

- Although a path has a fixed position when constructed, we can change the position later. This is called *translation* — moving the path without changing it orientation.
- We translate the current path by the *translatePath* method.

```
Kon.translatePath(100, 50); // (ii) translate the current path 100 right and 50 down
Kon.translatePath(Ap.rar(60, 150)); // (iii) translate the current path 150, 60 degrees
```



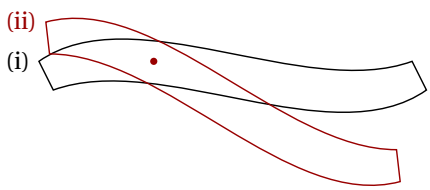
- We can generate and save multiple paths through translations.

```
Kon.pushPath(1);
Kon.translatePath(100, 50); Kon.pushPath(1);
Kon.translatePath(Ap.rar(60, 150)); Kon.pushPath(1);
```


Rotating Paths

- We can also rotate a path to a new position. This is called *rotation*, with an angle and a point to rotate about.
- We rotate the current path by the *rotatePath* method.

```
Kon.rotatePath(-20, 200, 100); // rotate the current path -20 degrees about (200,100)
```



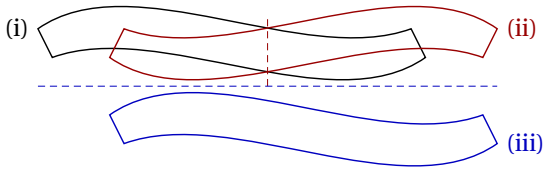
- We may add path rotation to our animation of the fill color.

```
Kon.rotatePath(-360/128.0, 200, 100);
```

Reflecting Paths

- With the ability to rotate, we need only to reflect horizontally and vertically.
- We reflect the current path by the *reflectPathH* and *reflectPathV* methods.

```
Kon.reflectPathH(400);  // (ii) reflect the current path horizontally about axis x = 400
Kon.reflectPathV(200);  // (iii) reflect the current path vertically about axis y = 200
```

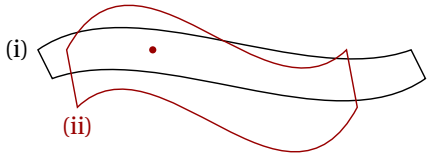


- For arbitrary reflection axes, we can first reflect and than rotate.

Scaling Paths

- Finally, we can scale a path to a different size. This is called *scaling*, with a factor and a center point of the scaling.
- We scale the current path by the *scalePath* method.

```
Kon.scalePath(200, 100, 0.75, 2);
    // scale the current path 0.75x and 2y about (200,100)
```



- We may also add path scaling to our animation of the fill color.

```
double cosa = Math.cos(s*2*Math.PI);
Kon.scalePath(200, 100, 1.5-cosa, 1.5-cosa);
```


Reading Homework

Textbook

- Section 14.11.

Internet

- Bézier curve (`http://en.wikipedia.org/wiki/B%C3%A9zier_curve`).
- Animated Bézier Curves (`http://www.jasondavies.com/animated-bezier/`).
- 2D computer graphics (`http://en.wikipedia.org/wiki/2D_computer_graphics`).
- sRGB (`http://en.wikipedia.org/wiki/Srgb`).
- Transformation matrix (`http://en.wikipedia.org/wiki/Transformation_matrix#Affine_transformations`).
- Matrix (mathematics) (`http://en.wikipedia.org/wiki/Matrix_(mathematics)`).