Lecture 05
Two Dimensional Graphics
Part 1

Graphics Overview

Working with Geometry

Reference: Java-Tutorials/tutorial-2015/2d/index.html

Why should you study this lecture?

- Nowadays, multimedia content is popular
- Graphics make a program more attractive
- Graphics decrease monotone of designed GUIs which contains text normally

Contents

- Graphics Overview
- Introduction to Java 2D Graphics
- 2D Graphics APIs
- How to Draw?
- Demonstrations

1- Graphics Overview

From Wikipedia

- **Pixel**, picture element: the smallest addressable element in an all points addressable display device; so it is the smallest controllable element of a picture represented on the screen
- Representing a pixel: <x, y, color data>
- **Common representation for color**: a triple <red, green, blue>. Value of each component in is in range of 0..255
- **Raster graphics image:** It is a dot matrix data structure representing a generally rectangular grid of pixels, or points of color, viewable via a monitor, paper, or other display medium. Raster images are stored in image files with varying formats (file.bmp, file.png, file.dib, …)

Graphics Overview…

- ***Vector graphics*** is the use of geometrical primitives such as points, lines, curves, and shapes or polygons—all of which are based on mathematical expressions—to represent images in computer graphics.
- Vector graphics are based on vectors (also called paths), which lead through locations called control points or nodes. Each of these points has a definite position on the x and y axes of the work plane and determines the direction of the path; further, each path may be assigned a stroke color, shape, thickness, and fill.
- These properties don't increase the size of vector graphics files in a substantial manner, as all information resides in the document's structure, which describes solely how the vector should be drawn.
- Vector graphics can be magnified infinitely without loss of quality, while pixel-based graphics cannot.

Graphics Overview…

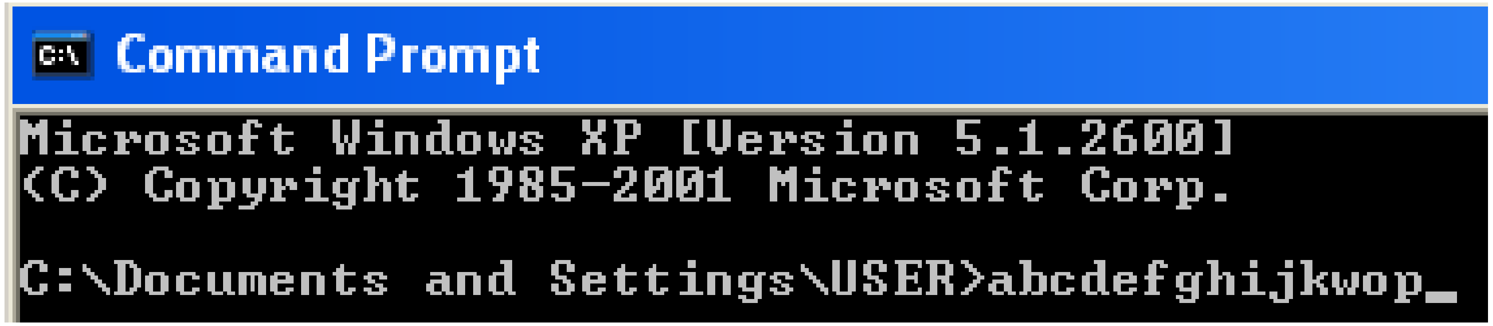
- ***Resolution***:
  - (theory) number of pixels per inch (2.54 cm)
  - (normal) rows*columns of a monitor
  - Common resolutions: 640x480, 800x600, 1024x768, 1280x800, 3200x2400, …
  - For more details: https://en.wikipedia.org/wiki/Graphics_display_resolution

Graphics Overview…

- ***Video card***: (also called a **video adapter**, **display card**, **graphics card**, **graphics board**, **display adapter**, **graphics adapter** or **frame buffer**) is an expansion card which generates a feed of output images to a display (such as a computer monitor)
- ***Graphics Accelerate Board***: a video card contains a graphical processing unit (GPU) to accelerate graphical presenting

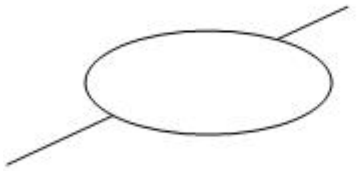
Graphics Overview…

- Video modes: Video card can work in two modes:
  - Text mode: The monitor is divided into a grid of large elements for presenting characters and each character is presented in one element. So, images can not be presented.
  - Graphic mode: The monitor is divided into a grid of small or tiny elements. So, images can be presented on it.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\USER>abcdefghijkwop_
```

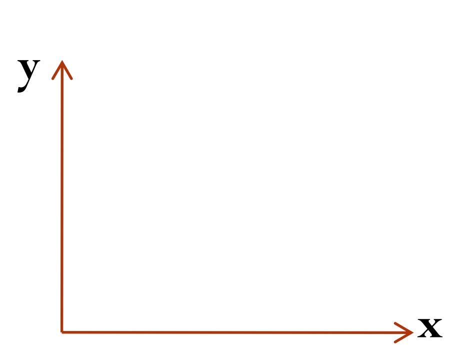
Text Mode

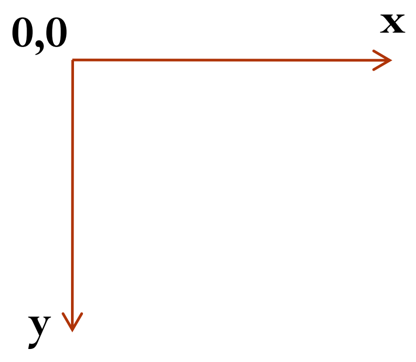


Graph Mode

Graphics Overview…

- Coordinates

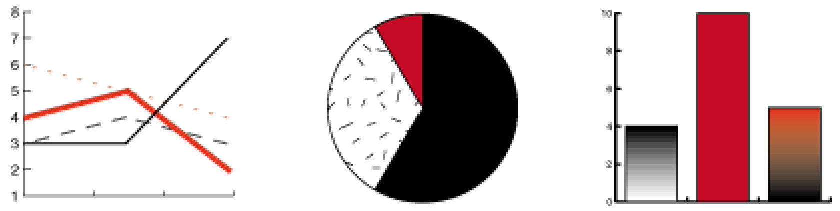




Common use co-ordinates Device co-ordinates

- Types of graphics:
  - Shapes: points, lines, shapes, polygons, colored shapes (vector graphics)
  - Photos (raster graphics)

2- Introduction to Java 2D Graphics



- Java 2D can:
  - Draw lines, rectangles and any other geometric shape.
  - Fill those shapes with solid colors or gradients and textures.
  - Draw text with options for fine control over the font and rendering process.
  - Draw images, optionally applying filtering operations.
  - Apply operations such as compositing and transforming during any of the above rendering operations.

TRƯỜNG ĐẠI HỌC FPT

3- Java 2D Graphics APIs

Package java.awt: Basic classes for drawing

Interface Hierarchy

- Shape
- Stroke
- Transparency
  - Paint

- java.lang.Object
  - Color
    - SystemColor
  - Font
  - FontMetrics
  - GradientPaint
  - Graphics
    - Graphics2D
  - GraphicsConfigTemplate
  - GraphicsConfiguration
  - GraphicsDevice
  - GraphicsEnvironment
  - Image
  - ImageCapabilities
  - MultipleGradientPaint
    - LinearGradientPaint
    - RadialGradientPaint
  - PageAttributes
  - PageAttributes.ColorType
  - PageAttributes.MediaType
  - PageAttributes.OrientationRequestedType
  - PageAttributes.OriginType
  - PageAttributes.PrintQualityType
  - java.awt.geom.Point2D
    - Point
  - Polygon
  - PrintJob
  - Java.awt.geom.RectangularShape
    - java.awt.geom.Rectangle2D
      - Rectangle

The **Graphics, Graphics2D** classes are abstract
➔ we can not use it directly.

• Each class representing a component in the packages java.awt, javax.swing implemented the method ***getGrahics()*** to get the appropriate concrete graphic objects which associate with it. This component plays a role as a drawing frame.

• Casting a Graphics object to get a Graphics2D object.

Java 2D Graphics APIs…

Package java.awt.geom: concrete classes for geometry

- AffineTransform
- Area
- CubicCurve2D
  - CubicCurve2D.Double
  - CubicCurve2D.Float
- Dimension2D
- FlatteningPathIterator
- Line2D
  - Line2D.Double
  - Line2D.Float
- Path2D
  - Path2D.Double
  - Path2D.Float
    - GeneralPath
- Point2D
  - Point2D.Double
  - Point2D.Float

- Quadratic: A curve of the second degree
- Cubic Curves: A curve of the third degree

- QuadCurve2D
  - QuadCurve2D.Double
  - QuadCurve2D.Float
- RectangularShape
  - Arc2D
    - Arc2D.Double
    - Arc2D.Float
  - Ellipse2D
    - Ellipse2D.Double
    - Ellipse2D.Float
  - Rectangle2D
    - Rectangle2D.Double
    - Rectangle2D.Float
  - RoundRectangle2D
    - RoundRectangle2D.Double
    - RoundRectangle2D.Float

Graphics versus Graphics2D

```
void drawLine(int x1, int y1, int x2, int y2)
void drawOval(int x, int y, int width, int height)
void drawPolygon(int[] xPoints, int[] yPoints, int nPoints).
void drawPolygon(Polygon p)
void drawPolyline(int[] xPoints, int[] yPoints, int nPoints)
void drawRect(int x, int y, int width, int height)
void drawRoundRect(int x,int y,int width,int height,int arcWidth,int arcHeight)
```

The Graphics class has drawing methods based on values

```
void draw(Shape s)
void clip(Shape s)
void fill(Shape s)
void rotate(double theta)
void rotate(double theta, double x, double y)
void scale(double sx, double sy)
void shear(double shx, double shy)
void transform(AffineTransform Tx)
void translate(double tx, double ty)
void translate(int x, int y)
```

The Graphics class inherits methods from the Graphics class and drawing methods based on shape objects and special methods are implemented such as rotate (quay ảnh), scale(co dãn), shear(làm nghiêng) , transform (biến hình), translate (chuyển dịch điểm)

The java.awt.Color Class

- Static constants for basic colors: black, BLACK, red, RED, pink, PINK, orange, ORANGE, …
- Common Constructors:

Color(int r, int g, int b), range 0-255

Color(int r, int g, int b, int a), r, g, b, alpha: 0-255

Color(float r, float g, float b), range: 0.0 - 1.0

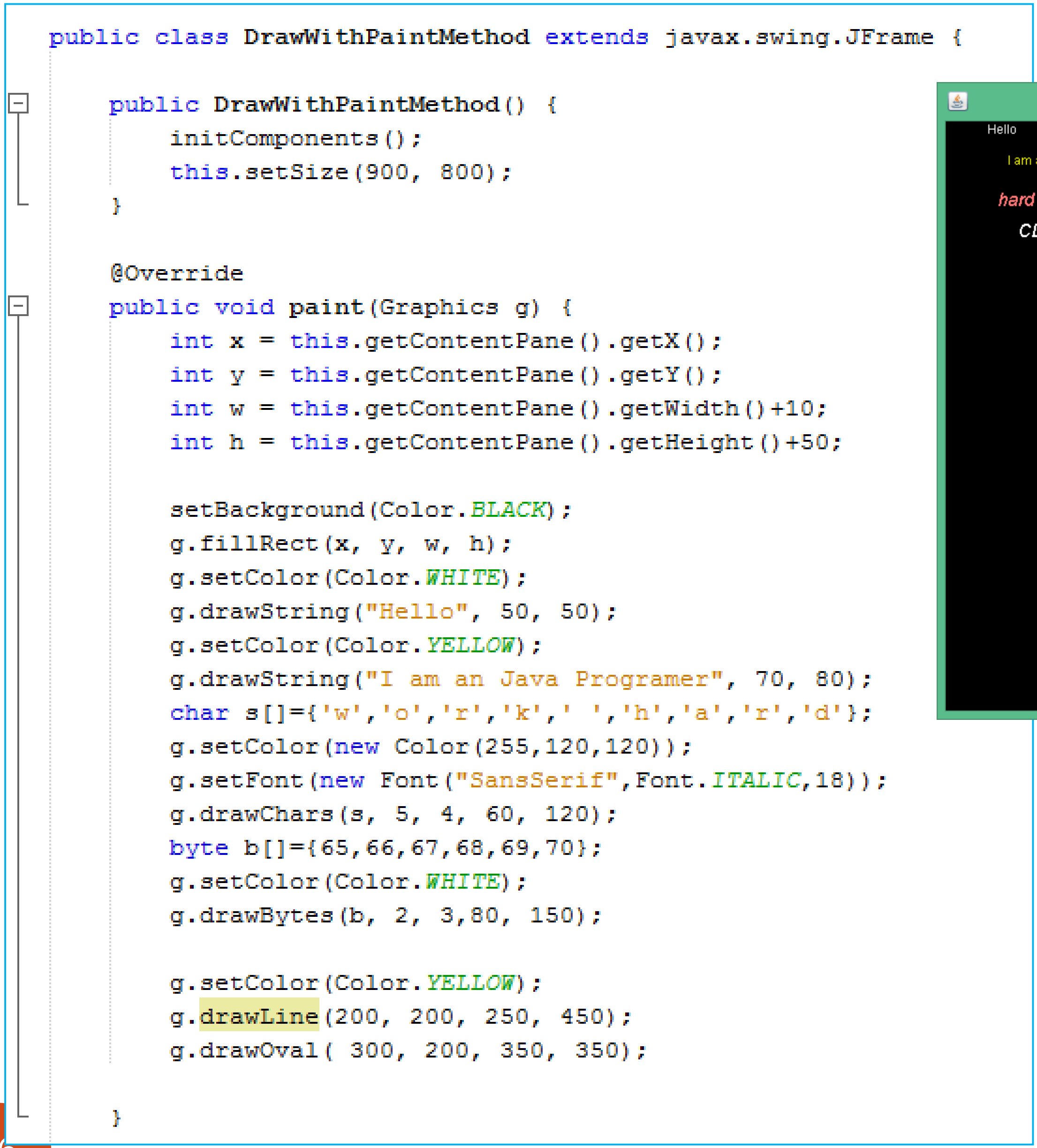
Color(float r, float g, float b, float a), range 0.0-1.0

Alpha: level of transparency

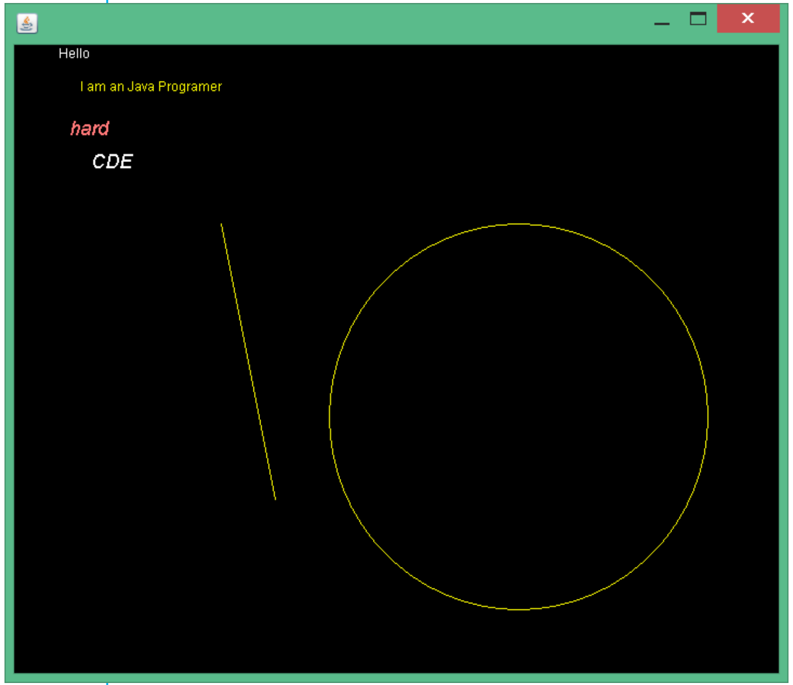
4- How to Draw?

- **Drawing as soon as GUIs appear:**
  - Overide the method ***paint****(Graphics g)*. This method will be called as default when the class is loaded to draw it.
  - Re-draw:
    - - Call the method ***update(Graphics g)*** or **repaint(void)**
- **Free Drawing:**
  - Graphics g = Component.getGraphics();
  - g.drawXXX(params)

Demo 1: Using the method paint(...)



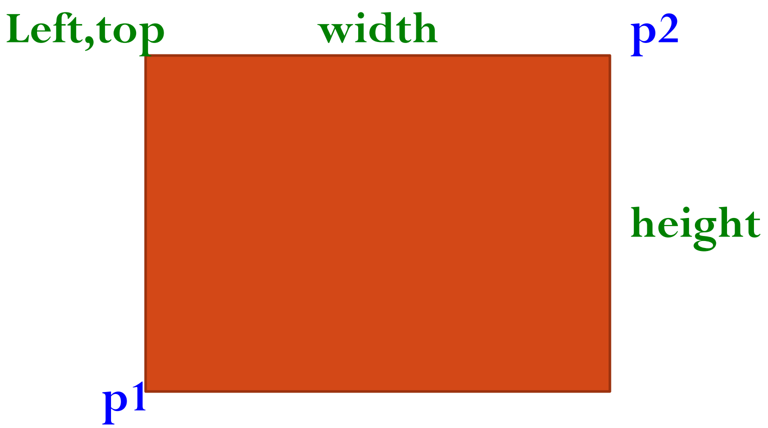
```
public class DrawWithPaintMethod extends javax.swing.JFrame {

    public DrawWithPaintMethod() {
        initComponents();
        this.setSize(900, 800);
    }

    @Override
    public void paint(Graphics g) {
        int x = this.getContentPane().getX();
        int y = this.getContentPane().getY();
        int w = this.getContentPane().getWidth()+10;
        int h = this.getContentPane().getHeight()+50;

        setBackground(Color.BLACK);
        g.fillRect(x, y, w, h);
        g.setColor(Color.WHITE);
        g.drawString("Hello", 50, 50);
        g.setColor(Color.YELLOW);
        g.drawString("I am an Java Programer", 70, 80);
        char s[]={'w','o','r','k',' ','h','a','r','d'};
        g.setColor(new Color(255,120,120));
        g.setFont(new Font("SansSerif",Font.ITALIC,18));
        g.drawChars(s, 5, 4, 60, 120);
        byte b[]={65,66,67,68,69,70};
        g.setColor(Color.WHITE);
        g.drawBytes(b, 2, 3,80, 150);

        g.setColor(Color.YELLOW);
        g.drawLine(200, 200, 250, 450);
        g.drawOval( 300, 200, 350, 350);

    }
```

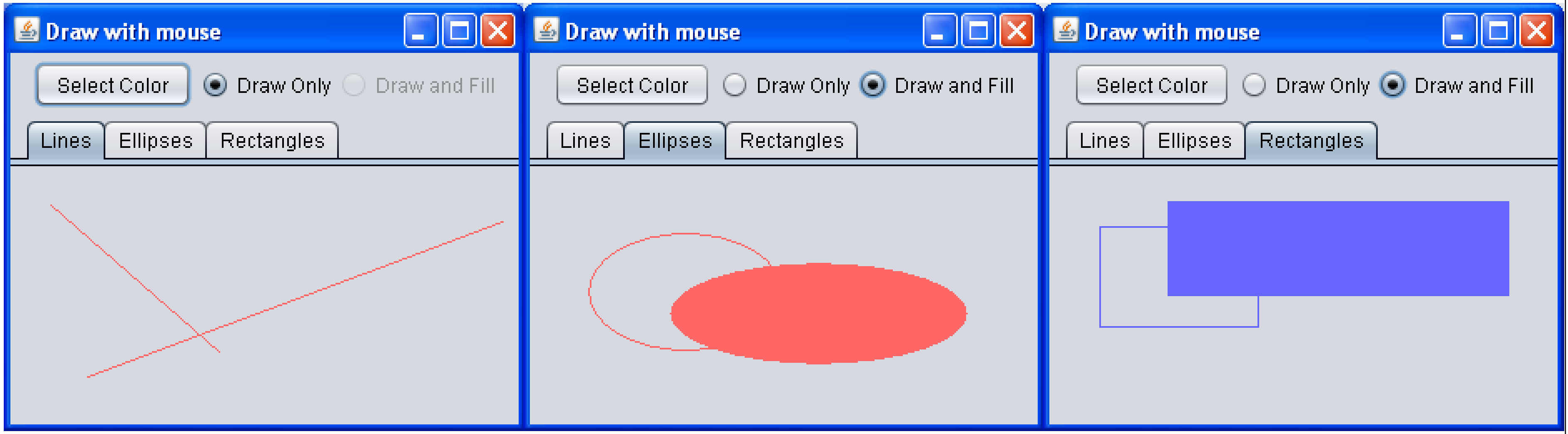


```
Hello
  I am an Java Programer
 hard
   CDE
```

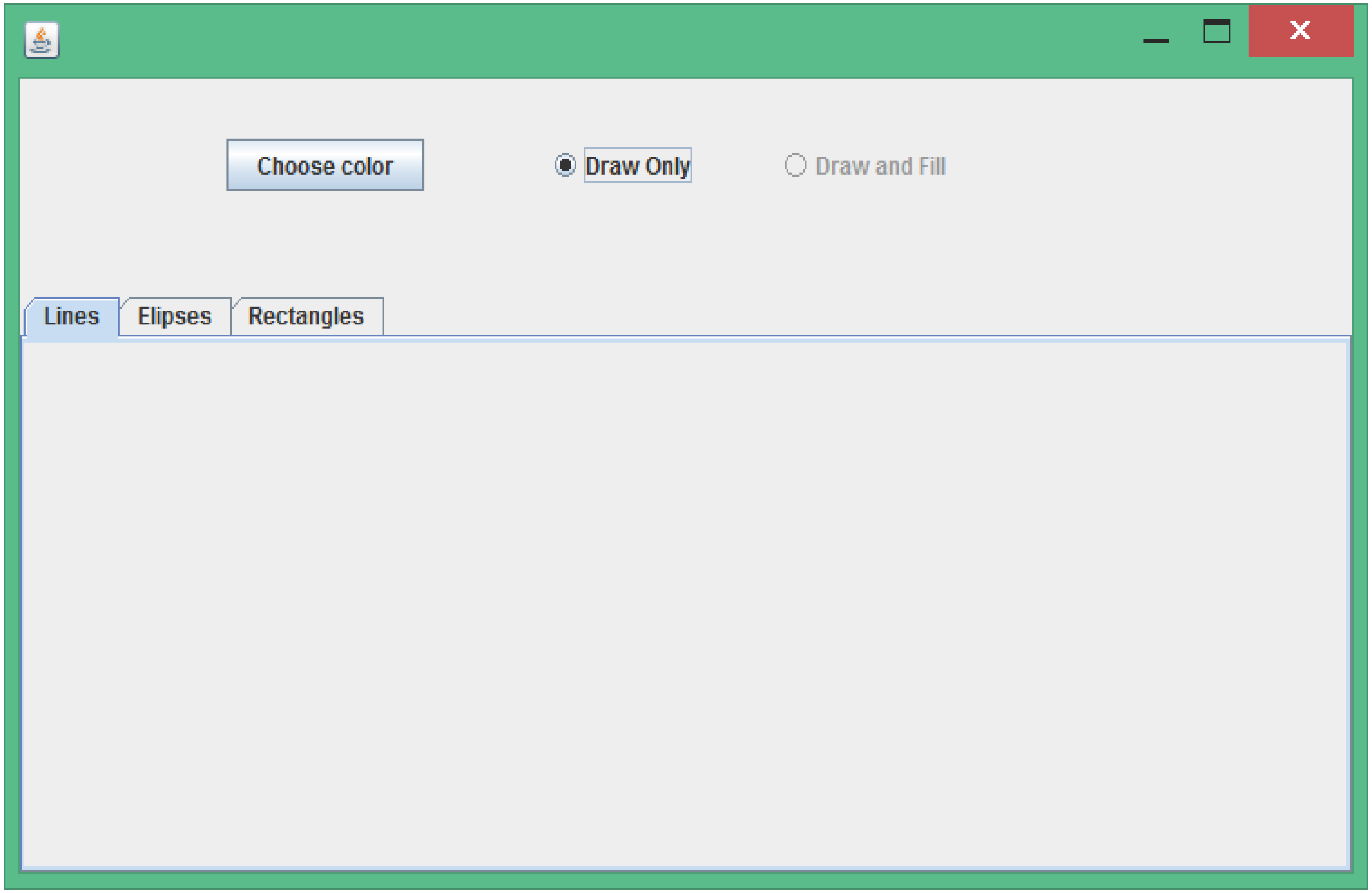
Demo 2: Drawing with Mouse

- The following demonstration will depict how to draw shapes using mouse events.
- A shape is determined by two points: p1 (when user presses the mouse) and p2 (when user releases the mouse).
- If an ellipse or a rectangle is chosen, from p1 and p2, the drawing area is determined appropriately.



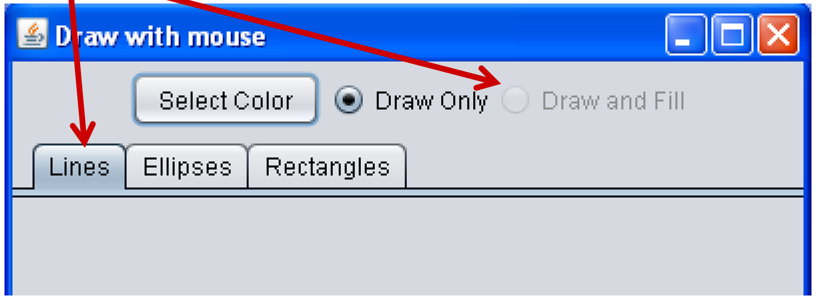


Demo 2: Design



Demo 2: Code

```
package graphs;
import java.awt.Graphics; // using graphics object
import java.awt.Point; // points for identifying drawing area
import java.awt.Color; // drawing color
import javax.swing.JColorChooser;
public class DrawWithMouse extends javax.swing.JFrame {
    Graphics g = null; // object for drawing
    Point p1=null, p2=null; // 2 points represent the drawing area
    Color color= Color.RED; // current color
    public DrawWithMouse() {
        initComponents();
        g= this.pLines.getGraphics();
        // when the program run, the panel pLines is default.
        // So, the graphic mode is draw only
        this.rdFill.setEnabled(false);
    }
```



TRƯỜNG ĐẠI HỌC FPT

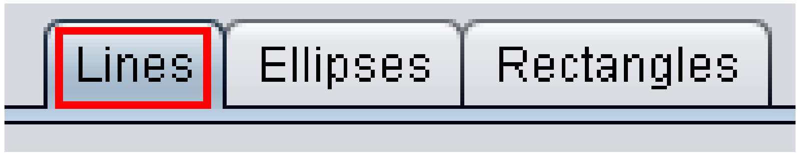
Demo 2: Code

```
private void btnColorActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    color= JColorChooser.showDialog(this, "Choose a color", Color.BLACK);
    g.setColor(color);
}
```

```
private void jTabbedPane1MouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    int index = this.jTabbedPane1.getSelectedIndex();
    switch (index){ // update the graphics object
        case 0: g= pLines.getGraphics();
                this.rdDraw.setSelected(true);
                this.rdFill.setEnabled(false);
                break;
        case 1: g= pEllipses.getGraphics();
                this.rdFill.setEnabled(true);
                break;
        case 2: g= pRectangle.getGraphics();
                this.rdFill.setEnabled(true);
                break;
    }
    g.setColor(color);
}
```

TRƯỜNG ĐẠI HỌC FPT

Demo 2: Code

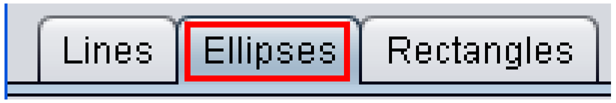


```
private void pLinesMousePressed(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    p1= evt.getPoint();// get the first point
}
```

```
private void pLinesMouseReleased(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    p2= evt.getPoint(); // get the second point
    g.drawLine(p1.x, p1.y, p2.x, p2.y);
    p1=p2=null;
}
```

TRƯỜNG ĐẠI HỌC FPT

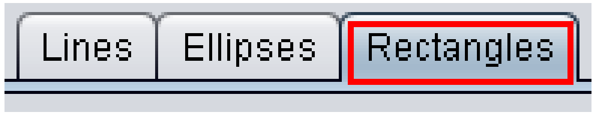
Demo 2: Code



```
private void pEllipsesMousePressed(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    p1= evt.getPoint(); // get the first point
}
```

```
private void pEllipsesMouseReleased(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    p2=evt.getPoint(); // get the second point
    int left= p1.x<p2.x? p1.x : p2.x; // determine the draw area
    int top= p1.y<p2.y? p1.y : p2.y;
    int width = p1.x- p2.x;
    if (width<0) width= -width;
    int height = p1.y- p2.y;
    if (height<0) height= -height;
    if (rdDraw.isSelected())
      g.drawOval(left, top, width, height);// draw
    else
      g.fillOval(left, top, width, height); // draw and fill
    p1=p2=null;
}
```

Demo 2: Code



```
private void pRectangleMousePressed(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    p1= evt.getPoint();// get the first point
}
```

```
private void pRectangleMouseReleased(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    p2= evt.getPoint();// get the second point
    int left= p1.x<p2.x? p1.x : p2.x; // determine the draw area
    int top= p1.y<p2.y? p1.y : p2.y;
    int width = p1.x- p2.x;
    if (width<0) width= -width;
    int height = p1.y - p2.y;
    if (height<0) height= -height;
    if (rdDraw.isSelected())
        g.drawRect(left, top, width, height);// draw
    else
        g.fillRect(left, top, width, height); // draw and fill
    p1=p2=null;
}
```

Summary

- Graphics Overview
- Introduction to Java 2D Graphics
- 2D Graphics APIs
- How to Draw?
- Demonstrations

Thank You