Programming 1

Lecture 13 – Java Graphics

Faculty of Information Technology
Hanoi University

Computer Graphics

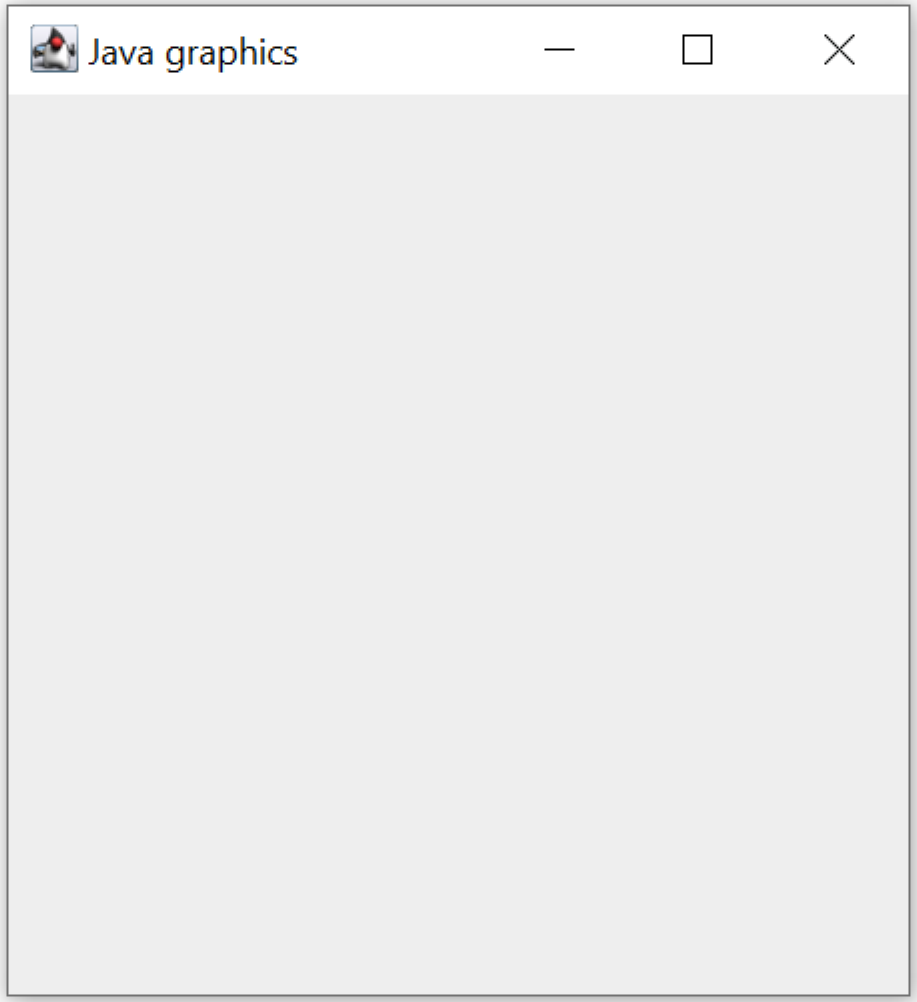
- Low level graphics components:
  - **Pixel:** a tiny dot on the computer screen
  - **Resolution:** the with and height of computer screen in pixels
    - 1920 x 1080: 2,073,600 pixels on your screen!
  - **Color:** each pixel has a color
    - A color is the mix of three numbers indicating the amount of red, green and blue (RGB)
    - `(0, 0, 0)` means BLACK, `(1.0, 0, 0)` means pure RED, `(1.0, 0.5, 0)` means orange (red + some green)

Computer Graphics

- High level graphics components:
  - **Image:** 2D matrix of pixels
  - **2D animation:** switching different images many times per second
  - **3D animation:** 2D pictures displayed in a manner which makes use of shadow, light and perception which appear to human eyes like 3D

javax.swing.JFrame

- **GUI** - Graphical User Interface
  - As opposed to *console* application
- A `JFrame` object can be used to create a window
  - Where other components can be added



javax.swing.JPanel

- A `JPanel` is an empty rectangle area which can be added to a `JFrame`
- Method: `paintComponent(Graphics g)`
  - Override this method to draw shapes, texts, images on the `JPanel`
  - A.k.a customizing the look of the `JPanel`
- We do not have to invoke this method
  - It executes automatically when a `JPanel` is displayed

java.awt.Graphics

- Contains methods for drawing lines, shapes, texts, images
- The `paintComponent()` method of `JPanel` receives a `Graphics` object as input
  - This `Graphics` object is provided for the `paintComponent()` method by Java

How to override `paintComponent()`?

- Create a class which `extends JPanel`
  - Such class is a *subclass* of `JPanel`
  - Now, `JPanel` is called its *superclass*
- Declare the `paintComponent(Graphics g)` method in the subclass
  - In a subclass, if you declare a method which has the same signature with a method in its superclass, you *override* the method

A subclass of `JPanel`

```java
import javax.swing.JPanel;
import java.awt.Graphics;

public class MyPanel extends JPanel {
    @Override
    public void paintComponent(Graphics g) {

    }
}
```

A JFrame containing a custom JPanel

```java
import javax.swing.*;
import java.awt.*;

public class JFrameDemo {
    public static void main(String[] args) {
        JFrame f = new JFrame("JFrame demo");
        JPanel p = new MyPanel();
        f.setSize(480, 360);
        f.add(p, BorderLayout.CENTER);
        f.setLocationRelativeTo(null); // center the frame
        // make the program terminate when window closes
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true); // make the window visible
    }
}
```

java.awt.Color

- Constructor:
  - `Color(float r, float g, float b)`
- Pre-defined colors:
  - `Color.BLACK`, `Color.WHITE`, `Color.RED`...
- This is enough for this lesson.

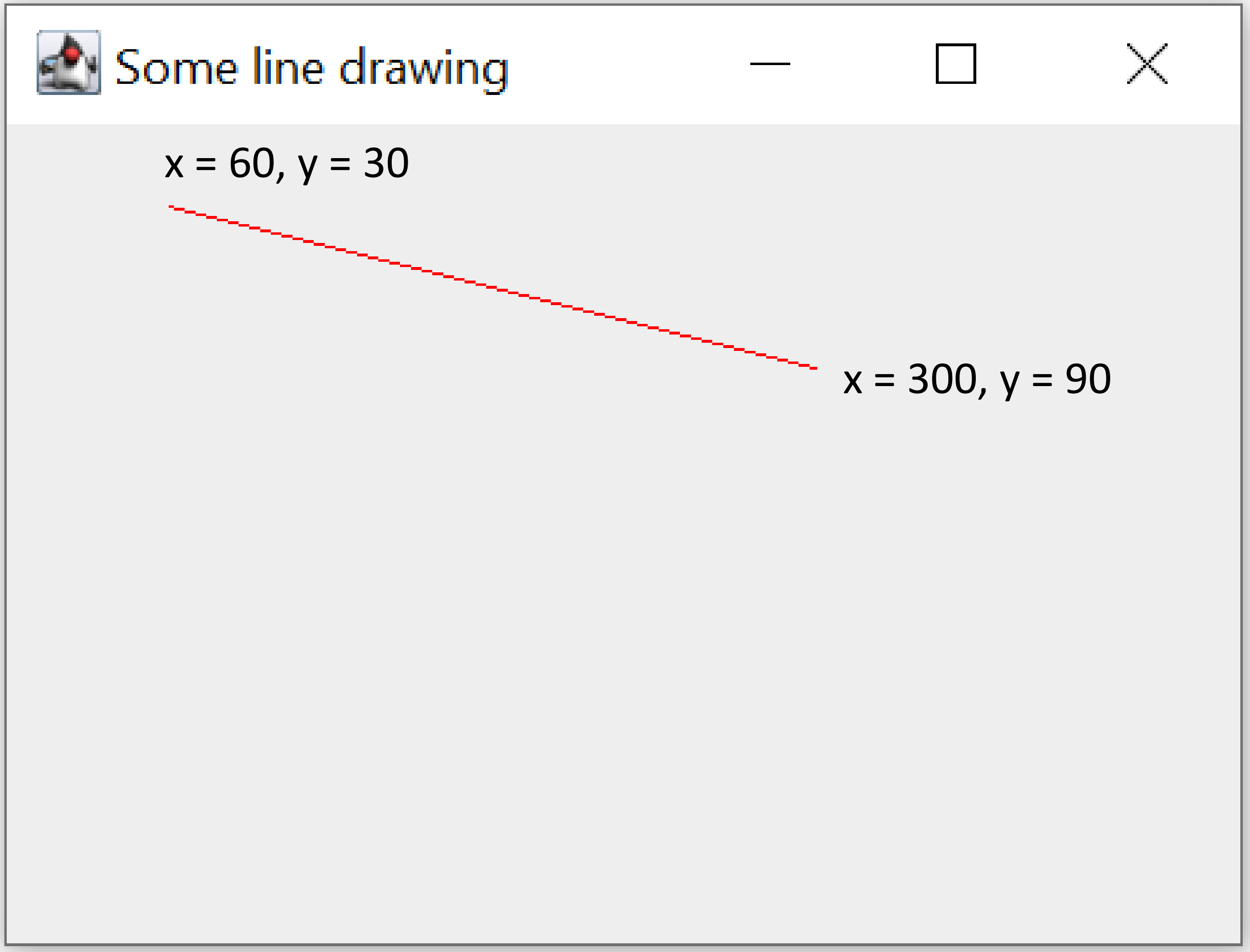
Simple Java drawing

- **Objective:** draw on the `JFrame` window
  - Solution: draw on the `JPanel`
  - How to?
    - Create subclass of `JPanel`
    - Write drawing codes in `paintComponent()`
- The input parameter of `paintComponent()` method (a `Graphics` object) can be used
  - The `Graphics` class provide many methods for drawing

Drawing lines

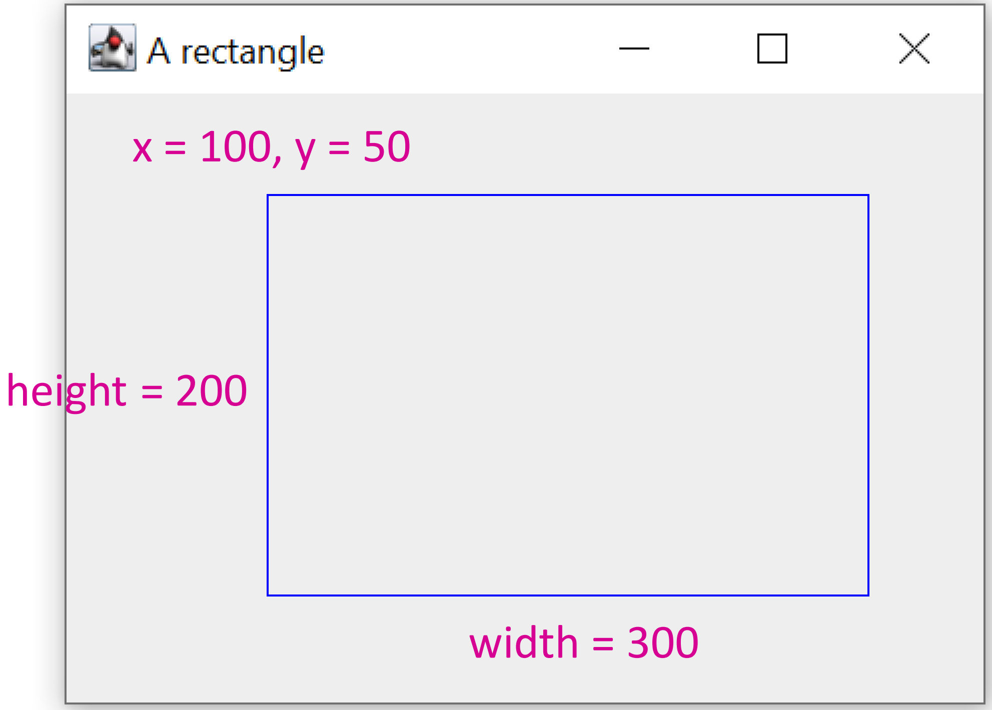
```
public class LinePanel extends JPanel {
    @Override
    public void paintComponent(Graphics g) {
        g.setColor(Color.RED);
        g.drawLine(60, 30, 300, 90);
    }
}

public class DrawLineDemo {
    public static void main(String[] args) {
        JFrame f = new JFrame("Some line drawing");
        JPanel p = new LinePanel();
        f.setSize(480, 360);
        f.add(p, BorderLayout.CENTER);
        f.setLocationRelativeTo(null);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
    }
}
```


Some line drawing
x = 60, y = 30
x = 300, y = 90

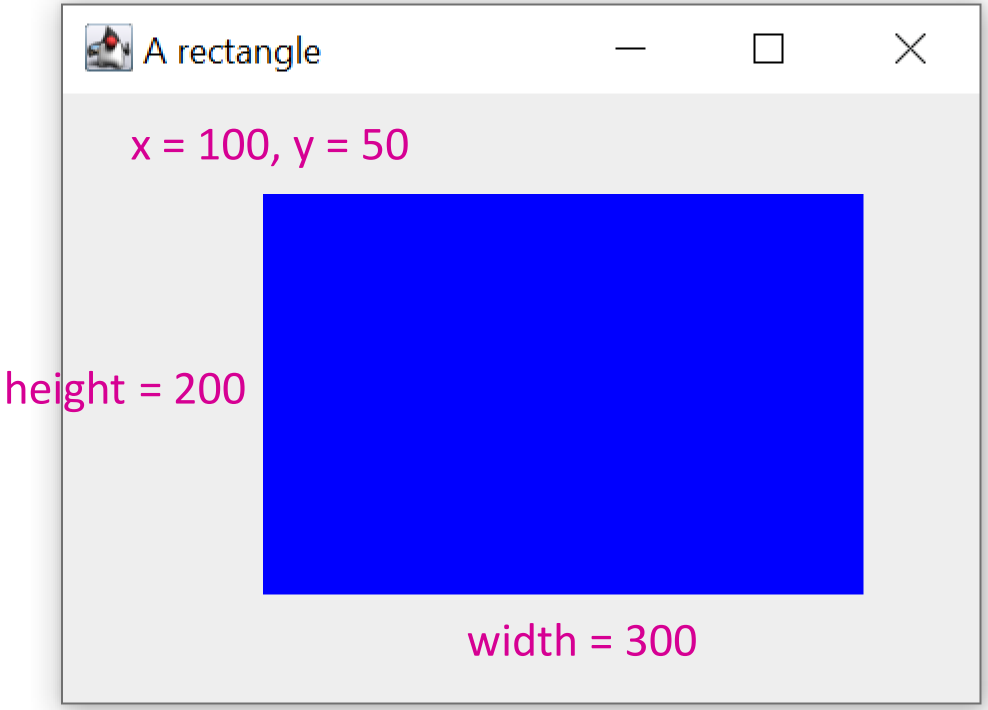
Drawing a rectangle

```java
public class RectPanel extends JPanel {
    @Override
    public void paintComponent(Graphics g) {
        g.setColor(Color.BLUE);
        g.drawRect(100, 50, 300, 200);
    }
}
```



Drawing a rectangle filled with color

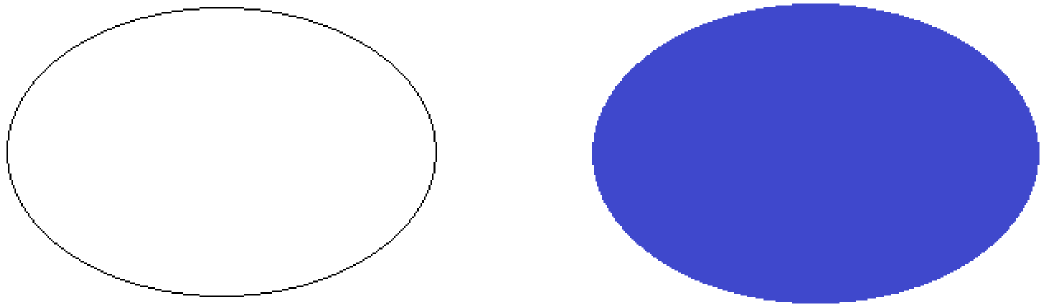
```
public class RectPanel extends JPanel {
    @Override
    public void paintComponent(Graphics g) {
        g.setColor(Color.BLUE);
        g.fillRect(100, 50, 300, 200);
    }
}
```



Ovals

- `drawOval(int x, int y, int width, int height)`
  - Used to draw the outline of an oval (which is a circle when `width` and `height` are equal)

- `fillOval(int x, int y, int width, int height)`
  - Fill version of `drawOval()` with the same parameters
  - Used to draw an oval which is filled with a color

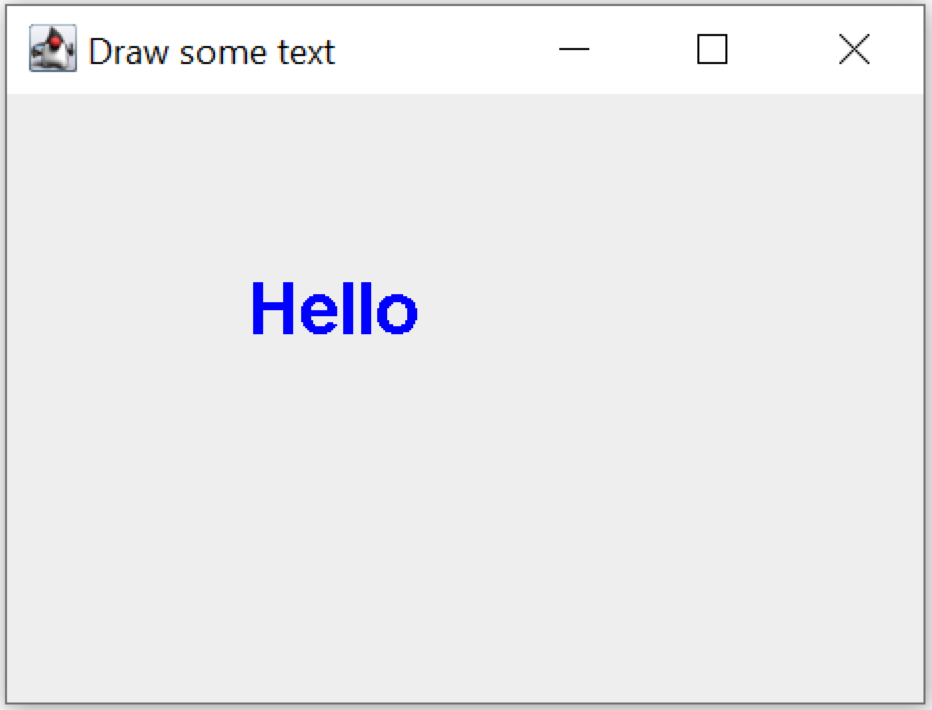


Triangles and other polygons

- `drawPolygon(int[] xPoints,`
  ```
              int[] yPoints, int nPoints)
  ```
- `drawPolyline(int[] xPoints,`
  ```
              int[] yPoints, int nPoints)
  ```

Drawing texts

- `setFont(Font font)`
- `drawString(String str, int x, int y)`
  - Draws the given text using current font and color.



Clear drawings

- `clearRect(int x, int y,`
```
          int width, int height)
```

- Clear the entire `JPanel`:

```
g.clearRect(0, 0, getWidth(), getHeight());
```

Working with images

- `ImageIO.read(File imgFile)`
  - Used to read an image file
  - Returns a `BufferedImage` object
- `Graphics.drawImage()`
  - Used to draw a `BufferedImage` (to a `JPanel`, for example
- `BufferedImage.getSubimage()`
  - Used to get a rectangle part of an image

Moving pictures (a.k.a. animation)

- `JPanel.repaint()`
  - Call this method to re-draw the `JPanel`
- `Thread.sleep(int millis)`
  - Use this method to delay the execution of the program
  - Used to control the frame rate of animation