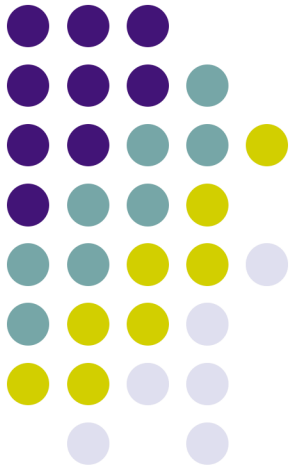
Principles of Computer Science I

Prof. Nadeem Abdul Hamid

CSC 120 – Fall 2006

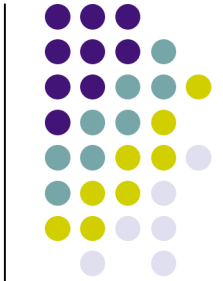
Lecture Unit 5 - Graphics



Lecture Outline

- Frame windows
- Drawing with shapes, colors, and text
- Programming applets
- Developing test cases

Frame Windows



- Graphical application (GUI = Graphical User Interface) shows information in a frame window
- To show a frame window in Java
  - Import `javax.swing.*` package
  - Construct JFrame object
  - Set its size, title, close behavior
  - Make it visible

Showing a Frame Window

EmptyFrameViewer.java

Showing a Frame Window

1. Construct a `JFrame` object

```
JFrame frame = new JFrame();
```

EmptyFrameViewer.java

Showing a Frame Window

1. Construct a `JFrame` object

```
JFrame frame = new JFrame();
```

2. Set frame size: width and height

```
frame.setSize(300, 400);
```

EmptyFrameViewer.java

Showing a Frame Window

1. Construct a `JFrame` object

```
JFrame frame = new JFrame();
```

2. Set frame size: width and height

```
frame.setSize(300, 400);
```

3. Set title of frame

```
frame.setTitle("An Empty Frame");
```

EmptyFrameViewer.java

Showing a Frame Window

1. Construct a `JFrame` object

```
JFrame frame = new JFrame();
```

2. Set frame size: width and height

```
frame.setSize(300, 400);
```

3. Set title of frame

```
frame.setTitle("An Empty Frame");
```

4. Set 'default close operation' (so that program exits when user closes the frame)

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

EmptyFrameViewer.java

Showing a Frame Window

1. Construct a `JFrame` object

```
JFrame frame = new JFrame();
```

2. Set frame size: width and height

```
frame.setSize(300, 400);
```

3. Set title of frame

```
frame.setTitle("An Empty Frame");
```

4. Set 'default close operation' (so that program exits when user closes the frame)

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

5. Make the frame visible

```
frame.setVisible(true);
```

EmptyFrameViewer.java

Drawing Shapes

- You do not draw directly on a frame
- To show anything in a frame (button, text, drawing, etc.) construct an appropriate *component* object and add it to the frame
- `JComponent` class represents blank component
- We *extend* the `JComponent` class to have it draw some shapes
  - Then add our modified version of `JComponent` to a frame to display the drawing

Extending JComponent

```
public class RectangleComponent extends JComponent {

    public void paintComponent(Graphics g) {
        // drawing instructions go here...

    } // end paintComponent method
} // end RectangleComponent class
```

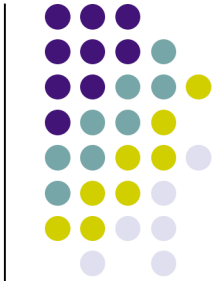
- `extends` keyword indicates that our class, `RectangleComponent`, inherits all the definitions and functionality of `JComponent`
  - But, we override the definition of the `paintComponent` method so that it does something we want
- `paintComponent` method is called by the Java system whenever the component needs to be redrawn

`paintComponent` Method

- Called automatically the first time window is shown
  - also called whenever window is resized or shown again after being hidden
- Takes a `Graphics` object parameter
  - A `Graphics` object stores the current graphics state: current color, font, background color, line size, etc.
- Swing toolkit provides `Graphics2D` class
  - `Graphics2D`: extended version of `Graphics` class that allows more sophisticated method to draw two-dimensional objects
  - To recover `Graphics2D` object from more primitive `Graphics` object, use a *cast*:

```
Graphics2D g2 = (Graphics2D) g;
```

Drawing with Graphics2D



```
public void paintComponent(Graphics g) {
      // Recover Graphics2D
      Graphics2D g2 = (Graphics2D) g;

      // Construct a rectangle and draw it
      Rectangle box = new Rectangle(5, 10, 20, 30);
      g2.draw(box);

      // Move rectangle 15 units to the right and 25 units down
      box.translate(15, 25);

      // Draw moved rectangle
      g2.draw(box);
}
```

- `Graphics` and `Graphics2D` classes part of `java.awt` package - needs to be imported
  - AWT = Abstract Windowing Toolkit

Complete RectangleComponent

```
import java.awt.*;      // imports all classes in java.awt package
import javax.swing.*;

/**
   A component that draws two rectangles.
*/
public class RectangleComponent extends JComponent
{
   public void paintComponent(Graphics g)
   {
      // Recover Graphics2D
      Graphics2D g2 = (Graphics2D) g;

      // Construct a rectangle and draw it
      Rectangle box = new Rectangle(5, 10, 20, 30);
      g2.draw(box);

      // Move rectangle 15 units to the right and 25 units down
      box.translate(15, 25);

      // Draw moved rectangle
      g2.draw(box);
   }
}
```

Complete RectangleComponent

```
import java.awt.*;      // imports all classes in java.awt package
import javax.swing.*;

/**
   A component that draws two rectangles.
*/
public class RectangleComponent extends JComponent
{
   public void paintComponent(Graphics g)
   {
      // Recover Graphics2D
      Graphics2D g2 = (Graphics2D) g;

      // Construct a rectangle and draw it
      Rectangle box = new Rectangle(5, 10, 20, 30);
      g2.draw(box);

      // Move rectangle 15 units to the right and 25 units down
      box.translate(15, 25);

      // Draw moved rectangle
      g2.draw(box);
   }
}
```

Displaying Rectangle Frame

- To display the RectangleComponent, you need to add it to a frame window

1. Construct frame as described earlier
2. Construct object of component class

```
RectangleComponent component = new RectangleComponent();
```

3. Add component to the frame

```
frame.add(component);
```

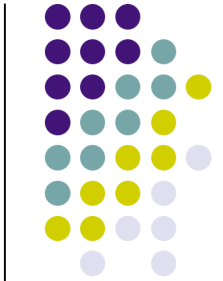
or `frame.getContentPane().add(component);` *in earlier Java versions*

4. Make frame visible

RectangleViewer Class

```
import javax.swing.JFrame;

public class RectangleViewer {

   public static void main(String[] args) {
      JFrame frame = new JFrame();

      final int FRAME_WIDTH = 300;
      final int FRAME_HEIGHT = 400;

      frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
      frame.setTitle("Two rectangles");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      RectangleComponent component = new RectangleComponent();
      frame.add(component);

      frame.setVisible(true);
   }
}
```

Applets



- “Mini-application” embedded in a web page
- Run with browser or applet viewer
- Differences (from applications)
  - Don’t have a `main` method
  - Embedded within HTML document (web page)
  - Subject to more security constraints
  - Not in control of own execution--respond to browser or viewer
- Can program graphics in single class - no need for separate component and viewer classes

Applet Code Skeleton

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

public class MyApplet extends JApplet {

    public void paint(Graphics g) {
        // Recover Graphics2D
        Graphics2D g2 = (Graphics2D) g;

        // Drawing instructions go here ...
    }

}
```

- Extend `JApplet` instead of `JComponent`
- Put drawing code inside `paint` method instead of `paintComponent` instead
- *RectangleApplet.java*

Running an Applet

- Need an HTML file with an `applet` tag
- HTML = Hypertext Markup Language
  - Plain text file with formatting commands - ‘tags’
  - Browser displays the contents of the file according to the formatting tags
  - To see HTML of file displayed in your browser, look for a ‘View Source’ command in the menus

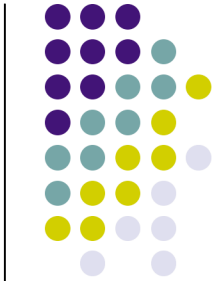
- Simplest file to display an applet:

```
<applet code="RectangleApplet.class" width="300" height="400">
</applet>
```

Another HTML File

```
<html>
   <head>
      <title>Two rectangles</title>
   </head>

   <body>
      <p>Here is my <i>first applet</i>:</p>

      <applet code="RectangleApplet.class"
              width="300" height="400">
      </applet>
   </body>
</html>
```

Viewing Applets



- Use applet viewer program with Java SDK

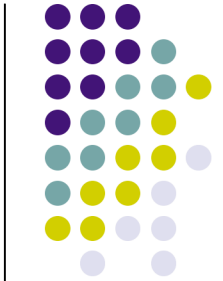
  `appletviewer RectangleApplet.html`

  - Only looks for <applet> tag in a web page-- allows you to test your applets before putting them on a web page

- Or use a 'Java 2-enabled' browser

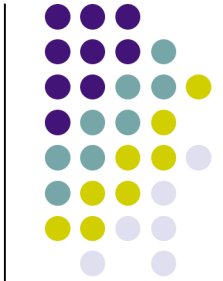
- WARNING: Browsers often save a copy of an applet in memory for a long time (e.g. until the entire application is exited) so if you change applet code and recompile, then reload the page in the browser, the browser may not use the latest version of the applet

The Internet



- 1960s - ARPANET
  - Universities, research institutions
  - Initial intent: allow remote execution of programs
  - ‘Killer app’: electronic mail
- 1972 - Internet (Bob Kahn)
  - Collection of interoperable networks
  - Share common *protocols* for transmitting data
  - 1983: TCP/IP: Kahn and Vinton Cerf
- 1989 - WWW (Tim Berners-Lee)
  - Hyperlinked documents
  - First interfaces clumsy to use
  - 1993: Mosaic graphical ‘web browser’ (Marc Andreesen)
- Lots of different protocols over Internet
  - FTP, telnet, gopher, file sharing, IM, smtp, …

Graphical Shapes

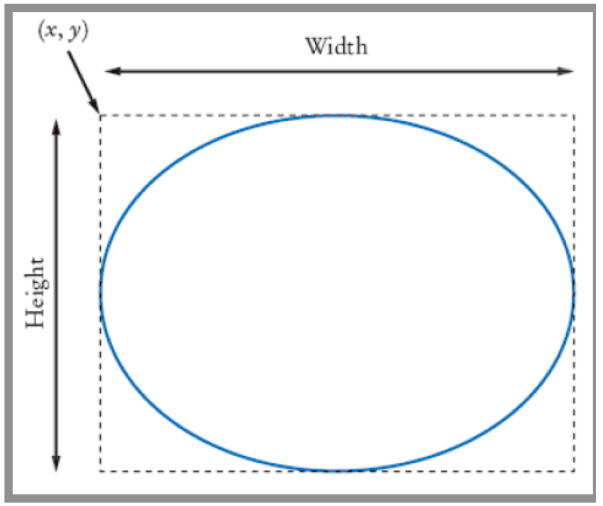


- Rectangles…
- Ellipses (and circles)
  - Use `Ellipse2D.Double` (or `Ellipse2D.Float`)
  - Stores coordinates as double values
- Lines
  - Use `Line2D.Double` (or `Line2D.Float`)
- Strange class names indicate they are *inner classes*
  - One class defined inside another
  - Just `import java.awt.geom.Ellipse2D` as usual and use above names

Drawing Ellipses and Lines

- Must **construct** and then **draw** shape objects

```
Ellipse2D.Double ellipse
  = new Ellipse2D.Double(x, y, width, height);
g2.draw(ellipse);
```



- To construct a line object

```
Line2D.Double segment
     = new Line2D.Double(x1, y1, x2, y2);
```

- or

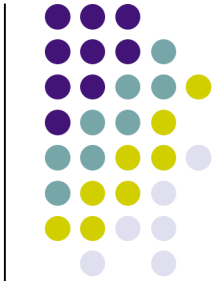
```
Point2D.Double from = new Point2D.Double(x1, y1);
Point2D.Double to = new Point2D.Double(x2, y2);
Line2D.Double segment = new Line2D.Double(from, to);

g2.draw(segment);
```

Drawing Strings

- Specify string and basepoint of first character

```
g2.drawString("Message", 50, 100);
```

Drawing Strings



- Specify string and basepoint of first character

```
g2.drawString("Message", 50, 100);
```

- Give code to draw a circle with center (100, 100) and radius 25
- Give code to draw a letter "V" by drawing two line segments

Colors

- Graphics context object keeps track of the current drawing color
- To change the color, supply a `Color` object to the `setColor` method
- Colors in Java specified by RGB (red-green-blue) model
  - Components given as `float` values (use an `F` suffix) between `0.0F` and `1.0F`

```
Color magenta = new Color(1.0F, 0.0F, 1.0F);
g2.setColor( magenta );
```

Predefined Colors and Fills

- `Color` class defines commonly used colors

  `Color.BLUE`, `Color.RED`, `Color.ORANGE`, ... (page 164)

- `setColor` method affects the line color
- To draw shapes filled in with current color

```
Rectangle box = ...;
g2.fill( box );
```

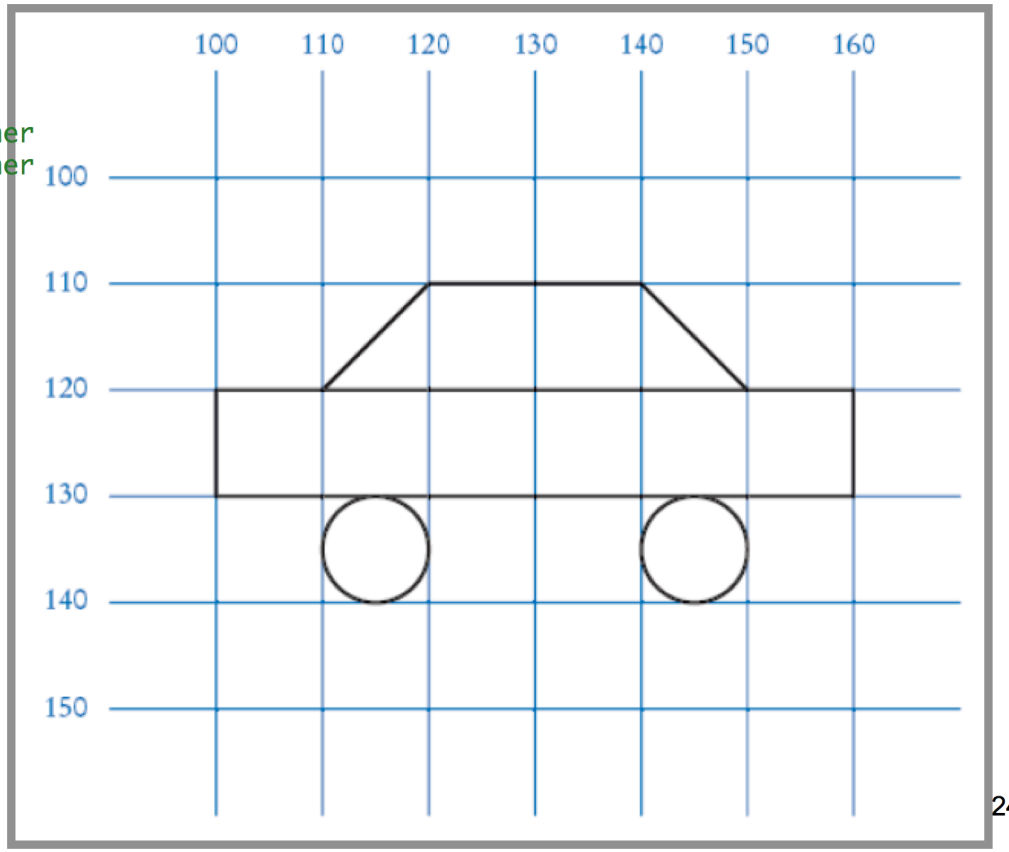
Drawing Complex Shapes

- Good practice: Make a class for each grapical object
- Plan complex shapes by making sketches on (graph) paper

```
public class Car {
    . . .

    public void draw(Graphics2D g2) {
        // Drawing instructions

        . . .
    }
}
```

Finding Shape Coordinates

```
/**
   A car shape that can be positioned anywhere on the screen.
*/
public class Car
{
   /**
      Constructs a car with a given top left corner
      @param x the x coordinate of the top left corner
      @param y the y coordinate of the top left corner
   */
   public Car(int x, int y)
   {

   }

   /**
      Draws the car.
      @param g2 the graphics context
   */
   public void draw(Graphics2D g2)
   {

   }
}
```



CarComponent Class

- Draw two cars – one in top left, one in bottom right corner of the component
- `JComponent` class provides `getWidth/getHeight` methods for dimensions of the component

- *Car.java*
- *CarComponent.java*

CarViewer

```
import javax.swing.JFrame;

public class CarViewer
{
   public static void main(String[] args)
   {
      JFrame frame = new JFrame();

      final int FRAME_WIDTH = 300;
      final int FRAME_HEIGHT = 400;

      frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
      frame.setTitle("Two cars");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      CarComponent component = new CarComponent();
      frame.add(component);

      frame.setVisible(true);
   }
}
```

- Try resizing the window

Object-Oriented Method of Programming Graphics

- Provide class for each object to be drawn
  - Implement a `draw` method, taking a `Graphics` parameter
- Provide (a simple) component class that calls the objects' `draw` methods
- Provide a viewer class that sets up a frame window

Reading Text Input

- Use JOptionPane

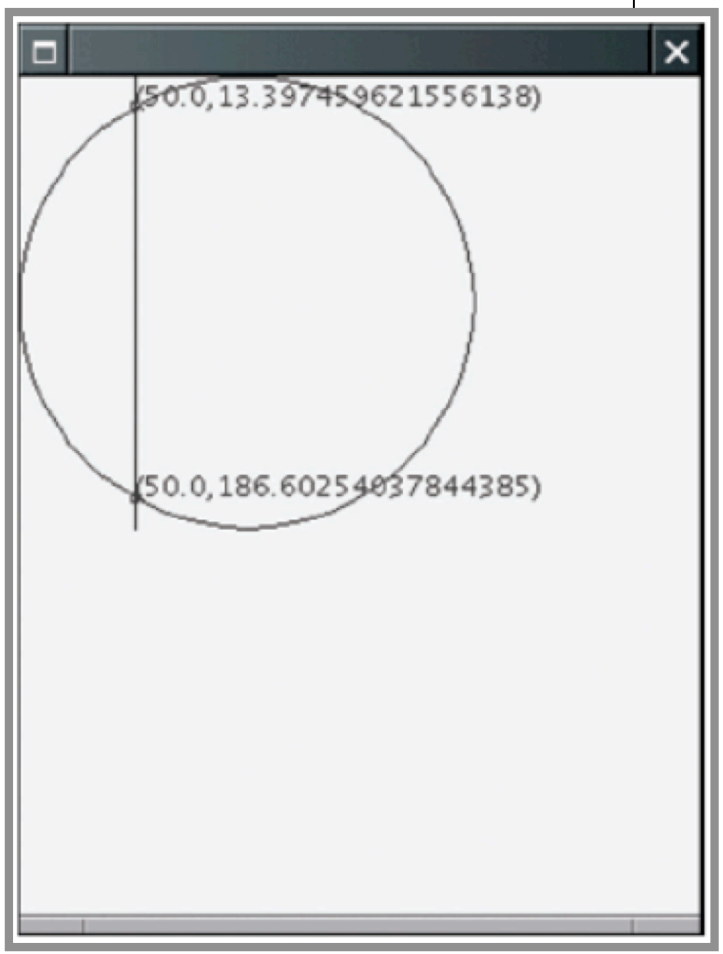
```
String input
          = JOptionPane.showInputDialog("Enter x");
double x = Double.parseDouble(input);
```

- In general, a poor choice for user interface but we will use it for now – much simpler than the alternative

- *ColorViewer.java*
- *ColoredSquareComponent.java*

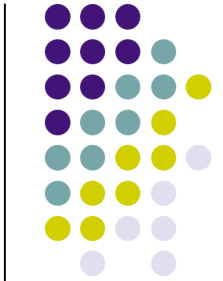
Comparing Visual and Numerical Information

- Compute intersection between circle and vertical line
- Circle has radius r = 100 and center (a, b) = (100, 100)
- Line has constant x value



- *IntersectionComponent.java*
- *LabeledPoint.java*
- *IntersectionViewer.java*

Using Test Cases



- Calculate test cases by hand to check your application
- Use special boundary conditions
  - $x = 0$; $x = 100$ …
- Use a few typical input values

- Don't be reluctant to double-check calculations by hand
- Random tinkering (switching + and - signs) is not a good way to debug a program