CompSci 230 S1 2018
Programming Techniques

A1 Help

Graphical User Interface (GUI) Implementing GUIs in Java

- The Java Foundation Classes (JFC) are a set of packages encompassing the following APIs:
  - AWT – Abstract Windows Toolkit (java.awt package)
    - The older version of the components
    - Rely on "peer architecture"…drawing done by the OS platform on which the application/applet is running
    - Considered to be "heavy-weight" components using native GUI system elements
  - Swing (Java 2, JDK 1.2+) (javax.swing package)
    - Newer version of the components
    - No "peer architecture"…components draw themselves
    - Most are considered to be "lightweight" that do not rely on the native GUI or OS

Graphical User Interface (GUI)
GUI elements

- **windows**: actual first-class citizens of desktop; also called top-level containers
  examples: frame, dialog box
- **components**: GUI widgets
  examples: button, text box, label
- **containers**: logical grouping for components
  example: panel

Graphical User Interface (GUI)
Swing component hierarchy

```
java.lang.Object
  +--java.awt.Component
        +--java.awt.Container
              |
              +--javax.swing.JComponent
              |     +--javax.swing.JButton
              |     +--javax.swing.JLabel
              |     +--javax.swing.JMenuBar
              |     +--javax.swing.JOptionPane
              |     +--javax.swing.JPanel
              |     +--javax.swing.JTextArea
              |     +--javax.swing.JTextField
              |
              +--java.awt.Window
                    +--java.awt.Frame
                          +--javax.swing.JFrame
```

Custom Painting

- Create an area for custom painting/drawing inside a JPanel
- Override the paintComponent method

```
public void paintComponent(Graphics g) {
  super.paintComponent(g);
```

  - Note: Call the superclass version of paintComponent as the first statement in the body of the overridden method to ensure that the component displays correctly.
- Note: We don't make a direct call to the paintComponent() method in our code.
  - This method is called **automatically** by the Java runtime whenever the JPanel area needs to be refreshed e.g.
    - when the JFrame is first created and displayed,
    - on some platforms the JPanel area is covered (the user moves to another application) and comes back to the JFrame,
    - when the user makes a change to the JFrame size.

Custom Painting
Graphics & Graphics2D

- Old graphics context: java.awt.Graphics
  - Used in Java 1.0 and 1.1, now obsolete
- New graphics context: java.awt.Graphics2D
  - Part of Java 2D (in Java 1.2 and later)
  - Although paintComponent() takes a Graphics object, what you get is really a Graphics2D!
- Basic methods for painting (Graphics and Graphics2D):
  - drawLine()
  - clearRect(), drawRect(), draw3DRect(), fillRect(), fill3DRect()
  - drawArc(), fillArc(), drawOval(), fillOval()
  - drawPolygon(), fillPolygon(), drawPolyLine()
  - drawString()

```
public void paintComponent(final Graphics g) { ...
  final Graphics2D g2d = (Graphics2D) g; // Just cast it…
  // Use g2d
}
```

Custom Painting
Java 2D

- Support for arbitrary shapes
  - A single draw() method, a single fill()
  - Draws or fills anything implementing
    - Line2D, Rectangle2D, RoundRectangle2D
    - Arc2D, Ellipse2D
    - QuadCurve2D, CubicCurve2D
    - ...
- Pen styles implement the Stroke interface (BasicStroke)
  - Different line widths, patterns, join styles
  - Use setStroke()
- Fill patterns implement the Paint interface
  - Color: Solid fill, default color space sRGB (rgb + alpha)
    - Color.RED, Color.GREEN, Color.BLACK, ...

```
Color cyan2 = new Color(0, 255, 255); // Between 0 and 255
```

    - TexturePaint: Tiles a picture (repeats as necessary)
    - GradientPaint: A gradient between two colors
  - Use setPaint() or the older setColor()

Custom Painting
Drawing Shapes

- In order to draw in the JPanel area we use the Graphics object.The Graphics object is supplied by the Java runtime as a parameter to the paintComponent() method.The Graphics class contains many instance methods:

```
drawLine(int x1, int y1, int x2 , int y2)
```

```
drawRect(int x, int y, int width, int height)
```

```
drawOval(int x, int y, int width, int height)
```

```
fillRect(int x, int y, int width, int height)
```

```
fillOval(int x, int y, int width, int height)
```

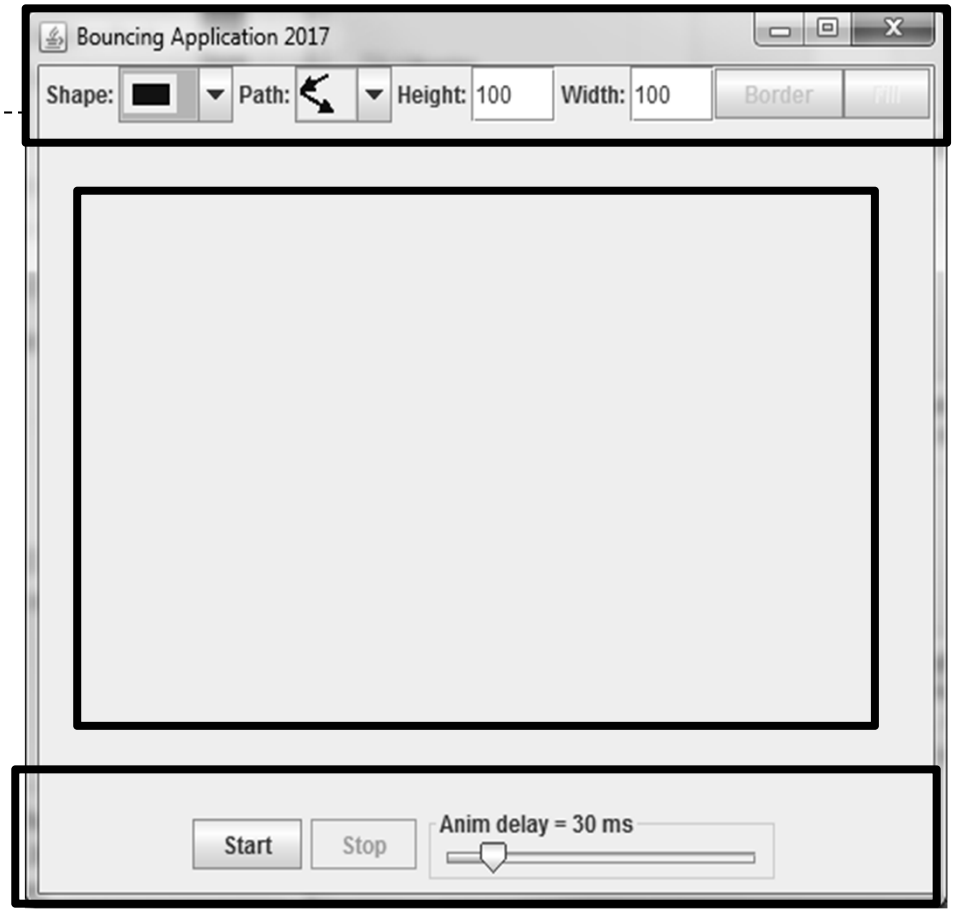
```
drawString(String text, int x, int y)
```

```
setColor(Color color)
```

```
g.setColor(Color.YELLOW);
```

A1 Class

- Layout:
  - Top: toolsPanel
    - Properties of shapes
  - Middle: AnimationPanel
    - Shapes bouncing area
  - Bottom: buttonPanel
    - Control the animation



- Note: You don't need to make any changes to the A1 class!

The Bouncing program - Background

- Animation:
  - animationThread.start()
  - execute run()
  - execute repaint()
  - execute paintComponent()
    - Loop through the shapes and execute the move() and draw() method
      - move() of the MovingRectangle
        - call path.move of a path
        - change the x and y position (i.e. top-left point)
        - Note: MovingPath is an Inner class of the shape, it can access and change the x, y coordinates
      - draw() of the MovingRectangle
        - call the draw method
        - draw the shape and handles if selected

```
public void paintComponent(Graphics g) {
   for (MovingShape currentShape: shapes) {
     currentShape.move();
     currentShape.draw(g);
     }
}
```

The Bouncing program - Background

- Adding a new shape
  - mouse click within the AnimationPanel area
  - Fire the mouseClicked event
    - If not selected
      - createNewShape(e.getX(), e.getY()) – at mouse point
        - Get all current values: shape, path, width, height …
        - Create a new instance and add it to the shapes array
    - If selected
      - Set the selected boolean to true

```
public void mouseClicked( MouseEvent e ) {
 ...
  if (!found)
    createNewShape(e.getX(), e.getY());
}
```

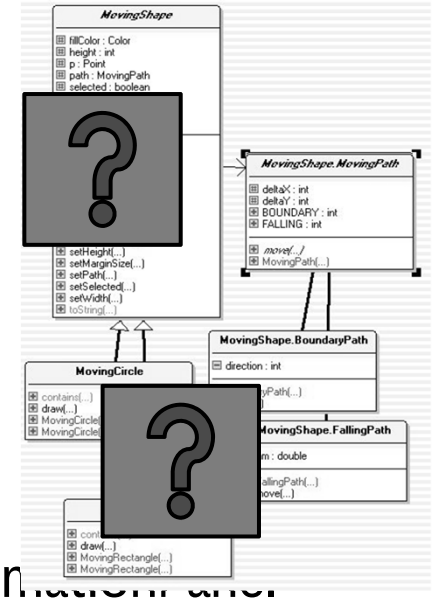
```
protected void createNewShape(int x, int y) {
...
shapes.add( new MovingRectangle(x, y, currentWidth, currentHeight, marginWidth,
marginHeight, currentBorderColor, currentFillColor, currentPath));
```

Tasks

- Task 1: MovingRectangle (5 marks)
- Task 2: MovingOval (5 marks)
- Task 3: MovingChecker (10 marks)
- Task 4: MovingGradient (10 marks)
- Task 5: MovingPattern (10 marks)
- Task 6: New Path (5 marks)
- Style & Comments (5 marks)

Task 1 - MovingRectangle

- Create a new Class
  - The class hierarchy should be developed sensibly and in accordance with good object-oriented programming practice.
    - Extends …
  - Implement TWO Constructors
  - draw():
    - draw a rectangle shape
  - contains():
    - check if a mouse point is within the rectangle
  - Add comments
  - Add a new case in the createNewShape() method in Animation.java.
- Check the following:
  - New shape is drawn with the current values.
  - Users should be able to change the properties of selected shapes.



Task 2 - MovingOval

- Create a new Class
  - The class hierarchy should be developed sensibly and in accordance with good object-oriented programming practice.
    - Extends …
  - Implement TWO constructors
  - draw():
    - paint a circle/ an ellipse
  - contains():
    - check if a mouse point is within the circle/ellipse
  - Add comments
  - Add a new case in the createNewShape() method in AnimationPanel
- Check the following:
  - New shape is drawn with the current values.
  - Users should be able to change the properties of selected shapes.

```
Point EndPt = new Point(x + width, y + height);
dx = (2 * mousePt.x - x - EndPt.x) / (double) width;
dy = (2 * mousePt.y - y - EndPt.y) / (double) height;
return dx * dx + dy * dy < 1.0;
```

Task 3: MovingChecker

- Create a new Class
  - The class hierarchy should be developed sensibly and in accordance with good object-oriented programming practice.
    - Extends …
    - Add two instance variables: xNumBlock, yNumBlock (random number 1 to 10)
  - Implement TWO constructors
  - Do you need to override the draw and/or contains?

  - Add a new case in the createNewShape() method in AnimationPanel
- Check the following
  - New shape is drawn with the current values.
  - Users should be able to change the properties of selected shapes.

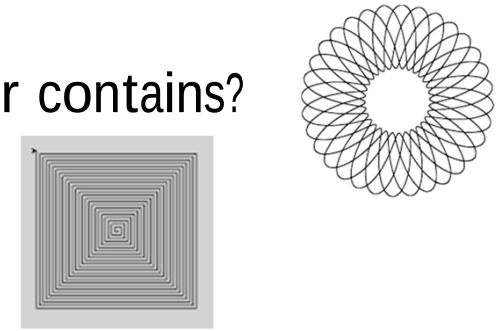
Task 4: MovingGradient

- Create a new Class
  - The class hierarchy should be developed sensibly and in accordance with good object-oriented programming practice.
    - Extends …
  - Implement TWO constructors
  - Do you need to override the draw and/or contains?

  - Add a new case in the createNewShape() method in AnimationPanel
- Check the following
  - New shape is drawn with the current values.
  - Users should be able to change the properties of selected shapes.

Task 5: MovingPattern

- Create a new Class
  - The class hierarchy should be developed sensibly and in accordance with good object-oriented programming practice.
    - Extends …
  - Implement TWO constructors
  - Do you need to override the draw and/or contains?



  - Add a new case in the createNewShape() method in AnimationPanel
- Check the following
  - New shape is drawn with the current values.
  - Users should be able to change the properties of selected shapes.
    - What will happen if the shape is a rectangle? Do you still get a nice pattern?

Task 6: New path

- Create a new Inner Class
  - The class hierarchy should be developed sensibly and in accordance with good object-oriented programming practice.
    - Extends …
  - Implement constructor
  - Override the move method
    - Add a formula to change the x and y coordinates
  - Add a new case in the setPath() method in MovingShape

```
public void move() {
    x = x + deltaX;
    y = y + deltaY;
    …
}
```

- Check the following:
  - New shape is bouncing using the new path.
  - Modify selected shapes to be bounced using the new path.