1.00 Tutorial 8

2D API, Model-View-Controller, Applets, Matrices & Linear Systems (1)

Today’s Schedule

- 2D API review
  - Exercise 1
- MVC discussion
  - Exercise 2
- Affine Transforms review
- Matrices & Linear Systems
- A brief note on Applets
- Problem Set 7 Discussion
  - Exercise 3

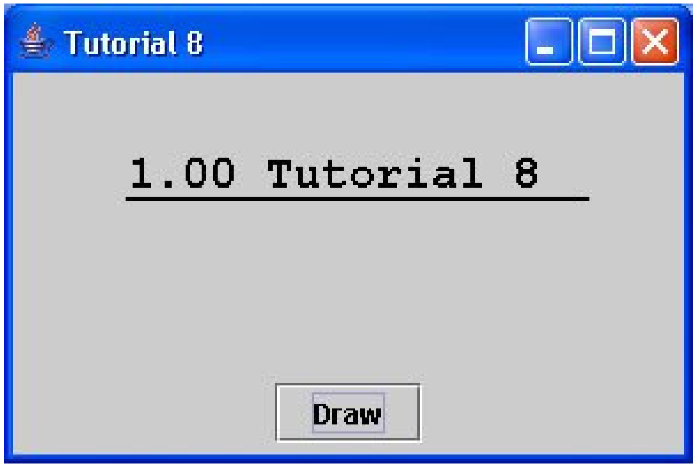
2D API
Exercise 1- Custom Drawing

- **Identify top level window / containers / component**

- **Where and what do we draw?**
  - Identify the things we need to draw
  - Where do we draw them?

- **List the methods of `Graphics2D` that we need for the exercise**

- **How de we set our own font?**



Source files called MyCanvas.java and Tutorial8.java

Exercise 1- Answers

- **Create a new class that extends `JPanel`**

  - Serve as a canvas for the custom drawing

- **Override `paintComponent()`**

  - `repaint()` calls `paintComponent()`

  - Don’t invoke `paintComponent()` explicitly

- **`Graphics2D` class methods**

  - `drawString()`

  - `draw()`

- **`Font` class**

Review - Custom Drawing

- Write a new class that extends **`JPanel`**
- Override **`paintComponent()`**

```
public class MyCanvas extends JPanel {
  ...

  public void paintComponent(Graphics g) {
      ...
  }
}
```

Source files called MyCanvas.java and Tutorial8.java

Review - First Things First

- **Invoke `super.paintComponent(g)`**
- **Cast `g` to a `Graphics2D` object**

```
public void paintComponent(Graphics g) {

  super.paintComponent(g);
  Graphics2D g2 = (Graphics2D)g;

  // Start drawing

}
```

Source files called MyCanvas.java and Tutorial8.java

Review - What Can We Draw?

- **String**

```
Font myFont = new Font(“Monospaced”, Font.BOLD, 12);
g2.setFont(myFont);
g2.drawString(“Draw This”, 100, 200);
```

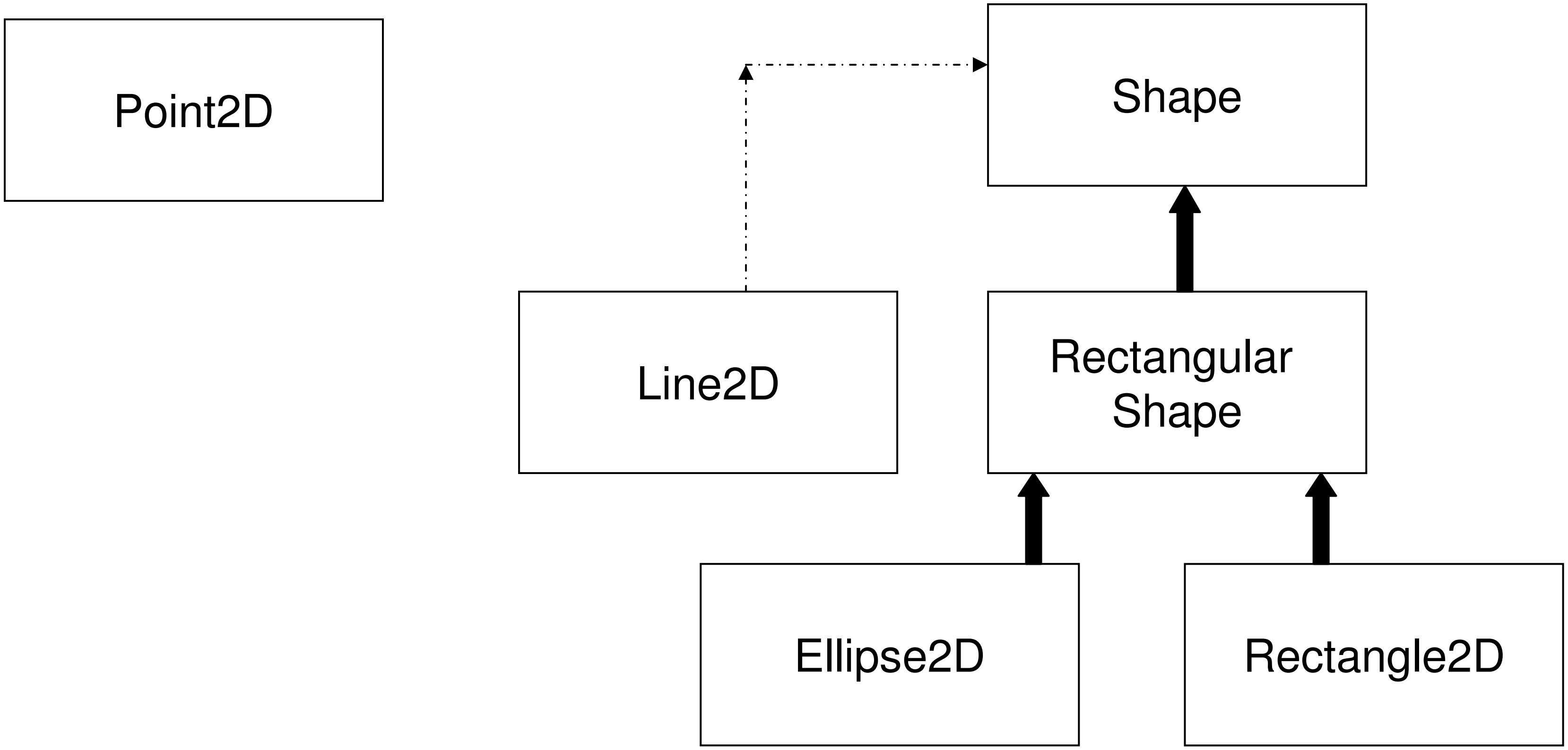
- **Shape (interface)**

- Known implementing classes:

`Line2D`, `Rectangle2D`, `Ellipse2D`

```
Shape s = new Rectangle2D.Double(10, 10, 20, 30);
Shape c = new Ellipse2D.Double(30, 40, 10, 10);
g2.draw(s);
g2.fill(c);
```

Review - What Can We Draw?



Model-View-Controller Paradigm

MVC programs are composed of 3 segments:

- the **View** manages the visible output (graphical / textual). Knows only about info *display* (ideally, has no domain knowledge).
- the **Model** models the domain of interest. It knows nothing about info display. Rather, it:
  - Responds to the View's requests for state
  - Responds to the Controller's requests to change state
- the **Controller** ties the Model and View together, instructing each to change as necessary in response to user actions and inputs.

Model-View-Controller in PS7

- Model-View-Controller paradigm separates responsibility:
  - The model (**`CatenaryModel`**) performs the catenary calculations from problem set 2. It contains no user interface code.
  - The view (**`CatenaryView`**) is the UI code. It draws lines, paints text, and in general displays a visual representation of the model. It contains a *reference* to the model.
  - The controller (**`CatenaryController`**) contains the event listeners: the code that runs when the user interacts with the program. It can modify the model and the view.

Exercise 2 – 2D API as MVC

Here we will

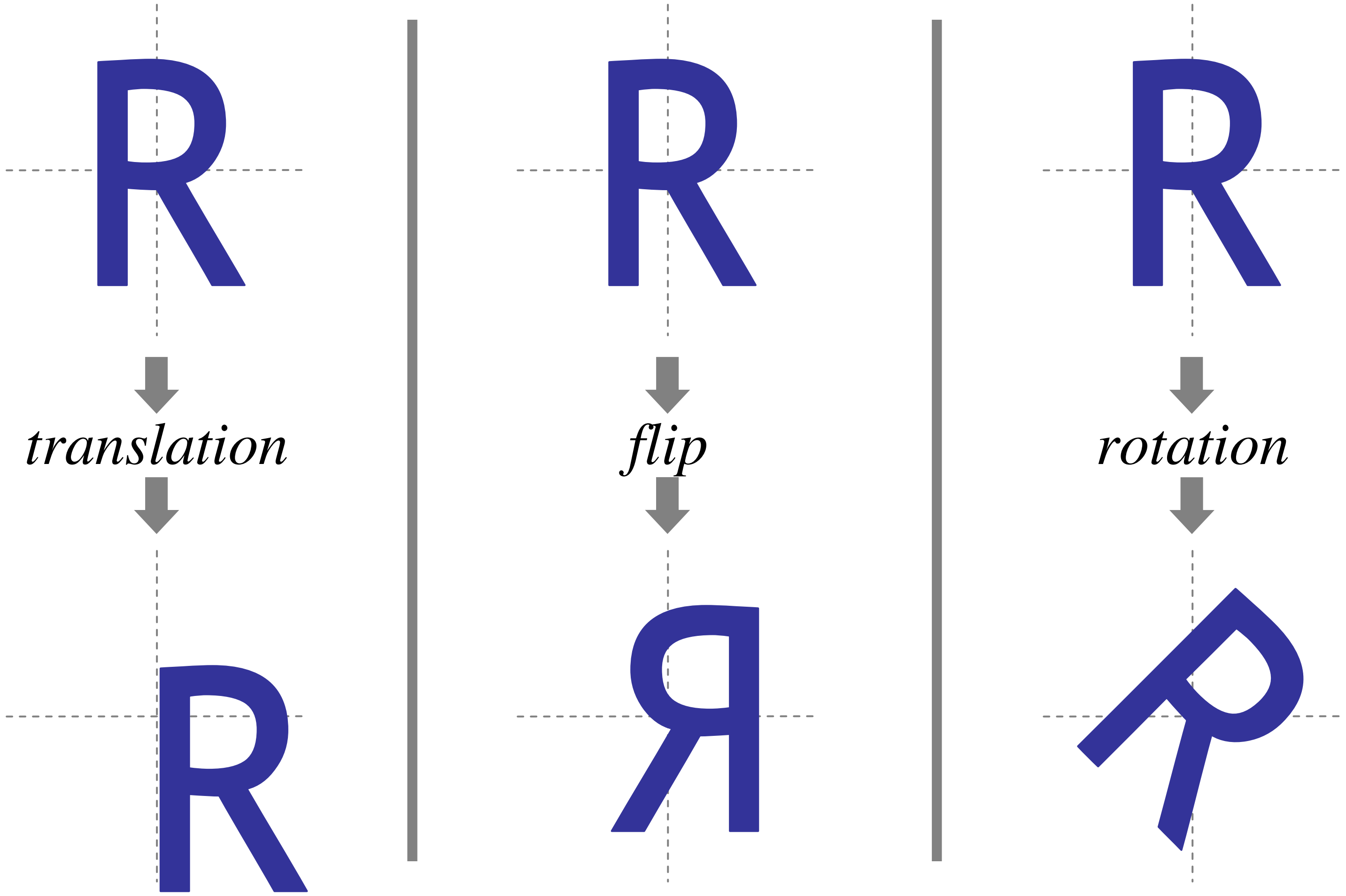
- Apply MVC to the example in Exercise 1
- Write code to
  - add a `CanvasModel` data member to the view (`MyCanvas2`) and the controller (`CanvasController`).
  - Add a `MyCanvas2` data member and a `JTextField` to the controller.
  - complete the button's anonymous `ActionListener` to instantiate both the model and the view and to use the string in the `JTextField` in the model
  - complete the view's `paintComponent()` so that the inverted string is printed 100 pixels below the original string

Affine Transformations

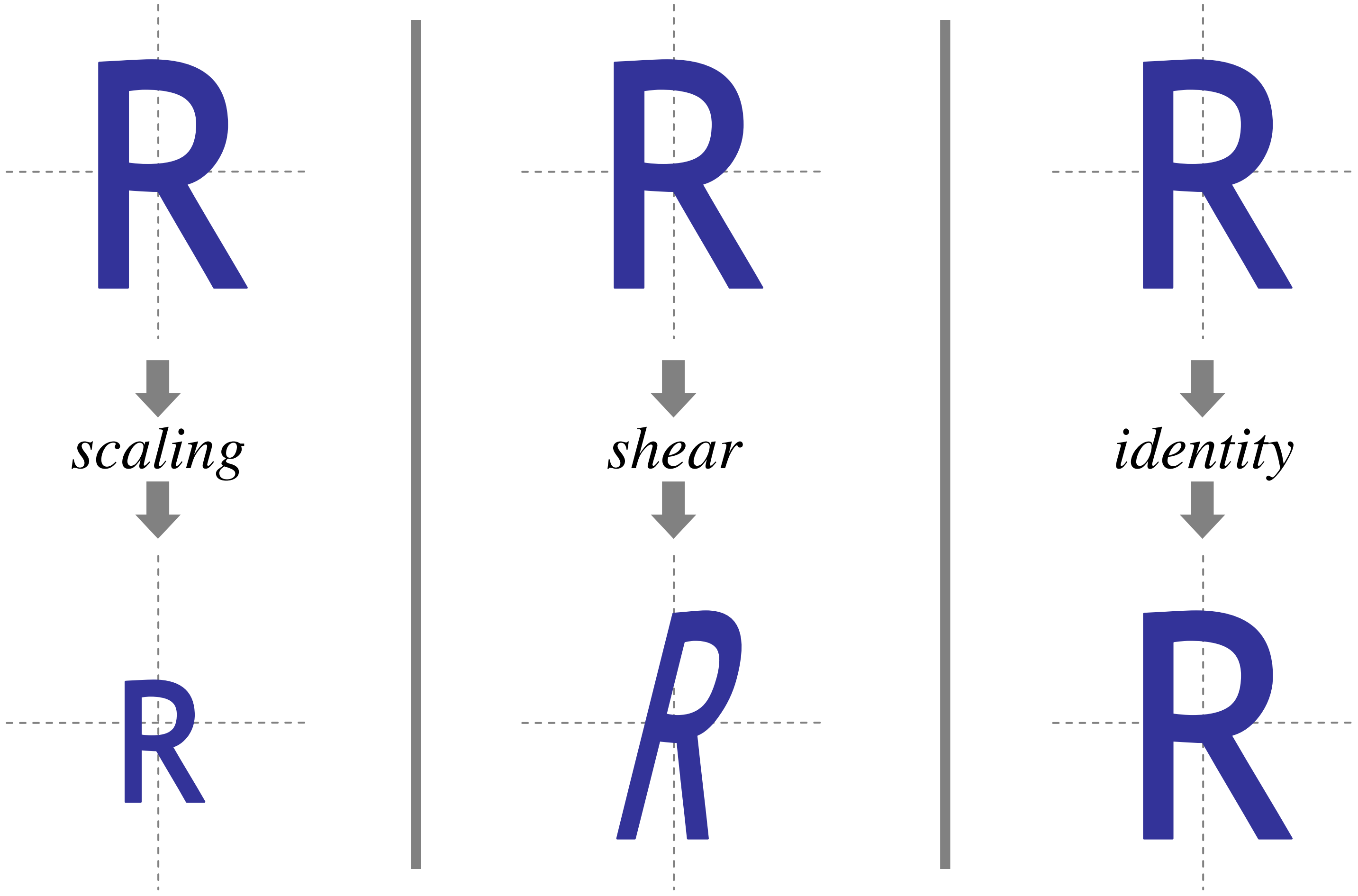
from Sun's Javadoc for `AffineTransform`

- A linear mapping from 2D coordinates to another set of 2D coordinates that preserves the "straightness" and "parallelness" of lines.
- Affine transformations can be constructed using sequences of translations, scales, flips, rotations, and shears.

Affine Transformations



Affine Transformations



scaling

shear

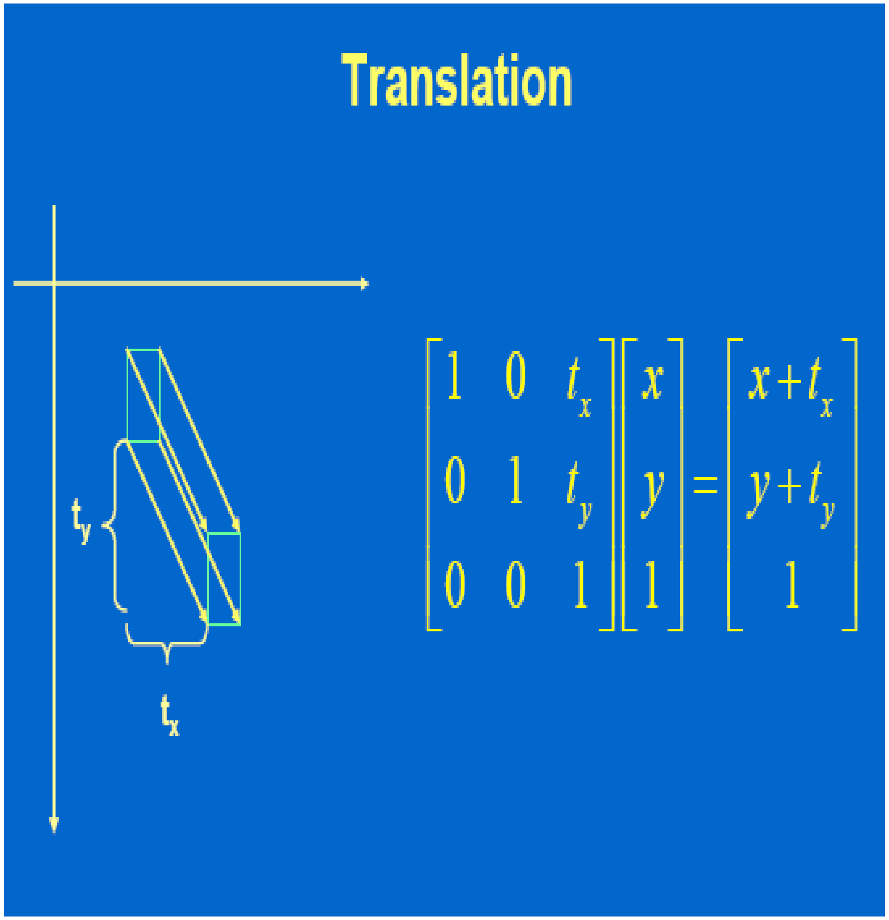
identity

Affine Transformations

- An Affine Transform simply encapsulates a 3 x 3 matrix for a given transformation.

- Approaches:

1. Apply AffineTransform using Graphics2D's **`transform`** method using (as seen in lecture).
2. You may also use AffineTransform's **`createTransformedShape`** method to create a new, transformed shape from an old one. Then you can draw the shape.



$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x+t_x \\ y+t_y \\ 1 \end{bmatrix}$$

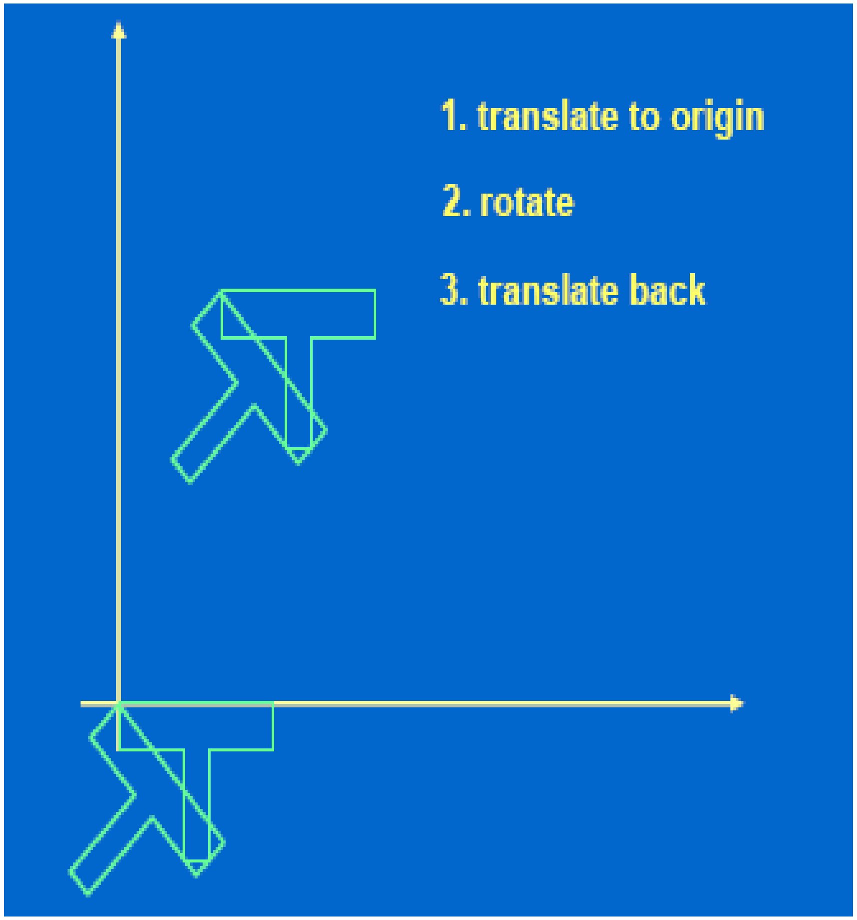
Source files called TranslatePanel.java, ScalePanel.java, RotatePanel.java and TransformMain.java

Affine Transformations - Review

- When we transform a shape, we transform each of the defining points of the shape, and then redraw it.

- If we scale or rotate a shape that is not anchored at the origin, it will translate as well.

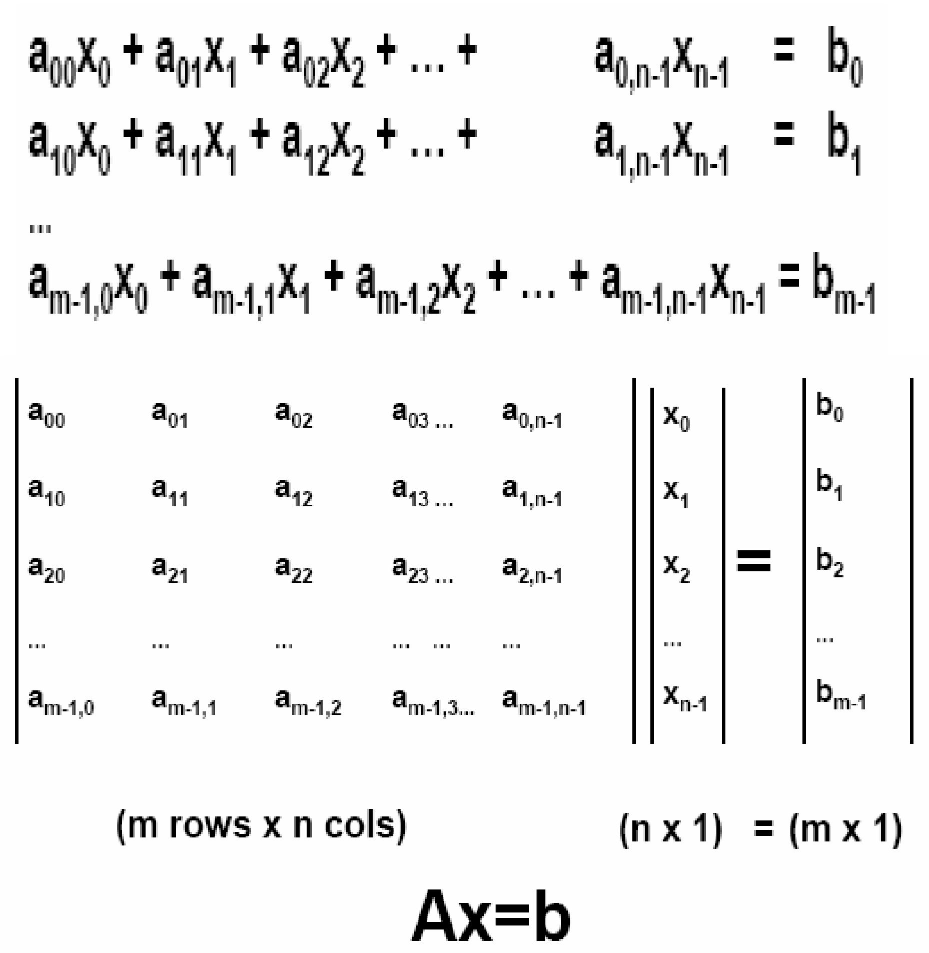
- If we just want to scale or rotate, then we should translate back to the origin, scale or rotate, and then translate back.



Source files called TranslatePanel.java, ScalePanel.java, RotatePanel.java and TransformMain.java

Matrices & Linear Systems (1)

- Matrices often used to represent a set of linear equations
- Coefficients a and right hand side b are known
- n unknowns x related to each other by m equations



$$a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + \ldots + \quad a_{0,n-1}x_{n-1} = b_0$$

$$a_{10}x_0 + a_{11}x_1 + a_{12}x_2 + \ldots + \quad a_{1,n-1}x_{n-1} = b_1$$

$$\ldots$$

$$a_{m-1,0}x_0 + a_{m-1,1}x_1 + a_{m-1,2}x_2 + \ldots + a_{m-1,n-1}x_{n-1} = b_{m-1}$$

$$\left|\begin{array}{ccccc} a_{00} & a_{01} & a_{02} & a_{03}\ldots & a_{0,n-1} \\ a_{10} & a_{11} & a_{12} & a_{13}\ldots & a_{1,n-1} \\ a_{20} & a_{21} & a_{22} & a_{23}\ldots & a_{2,n-1} \\ \ldots & \ldots & \ldots & \ldots\ \ldots & \ldots \\ a_{m-1,0} & a_{m-1,1} & a_{m-1,2} & a_{m-1,3}\ldots & a_{m-1,n-1} \end{array}\right| \left|\begin{array}{c} x_0 \\ x_1 \\ x_2 \\ \ldots \\ x_{n-1} \end{array}\right| = \left|\begin{array}{c} b_0 \\ b_1 \\ b_2 \\ \ldots \\ b_{m-1} \end{array}\right|$$

(m rows x n cols) (n x 1) = (m x 1)

Ax=b

Matrices & Linear Systems (1)

- If n=m, we will try to solve for unique set of x.
- Obstacles:
  - If any row (equation) or column (variables) is a linear combination of others, matrix is degenerate or not of full rank. No solution.
  - If rows or columns are nearly linear combinations, roundoff errors can make them linearly dependent. Failure to solve although solution might exist.
  - Roundoff errors can accumulate rapidly. While you may get a solution, when you substitute it into your equation system, you'll find it's not a solution.

- JAVA has 2D arrays for defining matrices. However, are no built-in methods for them

Applets

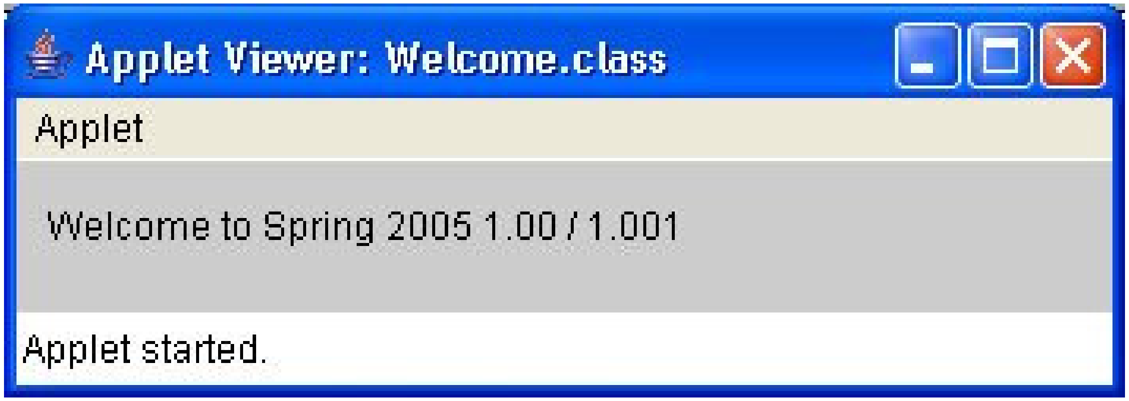
- Applets are programs embedded in web pages or run in an Applet viewer
- All applets are subclasses of the **`JApplet`** class
- Viewing Applets in Eclipse: **Run->Run As ->Java Applet**
- Recall lecture directions on converting Java Applications to Java Applets

A Sample Applet

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

public class Welcome extends JApplet {
   public void paint(Graphics g) {
        super.paint(g);
        Graphics2D g2 = (Graphics2D)g;
         g2.drawString("Welcome to Spring 2005 1.00 / 1.001", 10,
   25);
    }

}
```



```
Welcome to Spring 2005 1.00 / 1.001
```

```
Applet started.
```

Source files called Welcome.java and welcome.html

Applet Resources

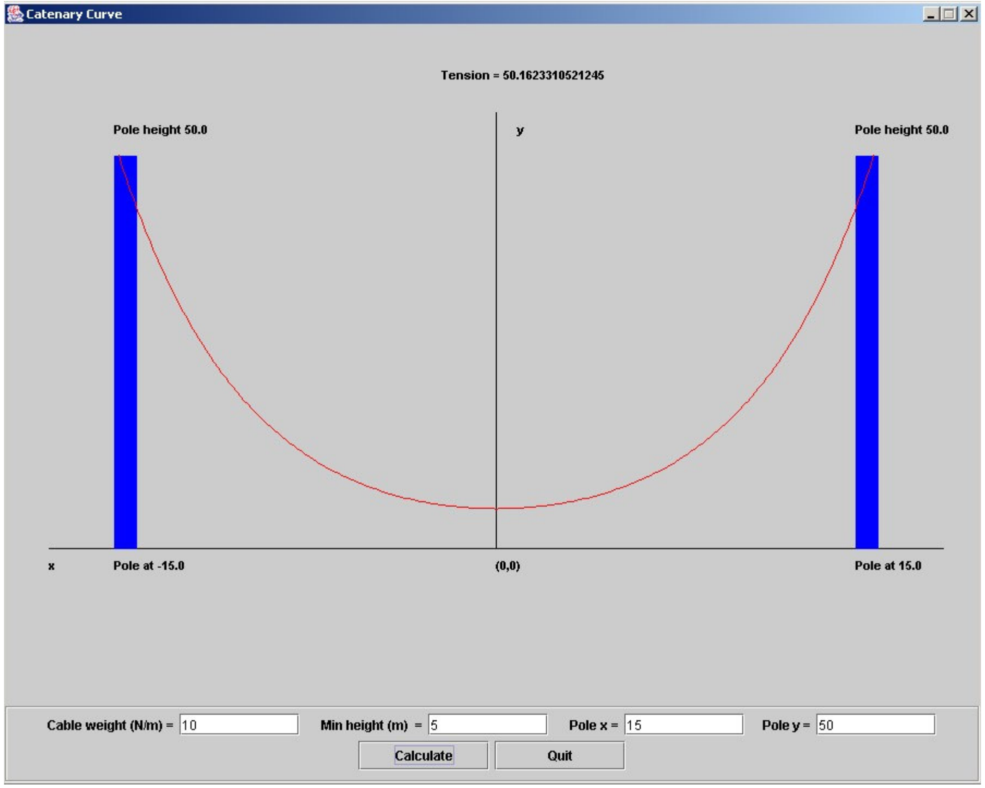
- The Java Tutorial : List of Applets http://java.sun.com/docs/books/tutorial/listofapplets.html
  - Tutorials on applets produced by Java for several releases of the program
  - Includes links to sample applets

- Applets (at java.sun.com) http://java.sun.com/applets/
  - A resource center for applet development
  - Links to sample code and applications.

- HTML Design at w3schools.com http://www.w3schools.com/html/default.asp
  - Free webpage development tutorials and resources

Homework 7 – UI for Catenary Height

- Continuation of Homework 6
  - Construct a Swing UI for Homework 2



Tension = 50.1623310521245

Cable weight (N/m) = 10

Min height (m) = 5

Pole x = 15

Pole y = 50

Homework 7 Continued

- Three pieces to the puzzle:
  - `CatenaryModel`: Does the number crunching for the model
  - `CatenaryView`: Inherits from `JPanel` and has methods for drawing the catenary, given a `CatenaryModel` instance
  - `CatenaryController`: Inherits from `JFrame` and has methods for creating the `CatenaryModel` and `CatenaryView` instances

Homework 7 Continued

- By now you must have implemented
  - `CatenaryModel`
    - Must be completely implemented
  - `CatenaryController`
    - Just the input and parsing components, creating instances of `CatenaryModel` and `CatenaryView`
- Now you need to implement
  - `CatenaryView`
    - Write constructor and data members (`CatenaryModel`)
    - Complete the `paintComponent`() method – draw the catenary, axes, text labels
  - `CatenaryController`
    - Complete the `actionListener`() for the Calculate button – create a new `CatenaryModel` or update the model parameters
    - Similarly, create a new `CatenaryView` instance or update the existing view

Homework 7 Continued – Exercise 3

- How do we draw the axes? (`CatenaryModel.paintComponent`)
  - Recall (0,0) is top left corner
  - The center of the view will vary as the window is resized.
- How do we make the "`Quit`" button functional
  - Must use anonymous inner class
  - Hint: how did we exit a program at the beginning of the semester?