Facial Recognition:

- France: Soon releasing nationwide facial recognition ID app
- China: Jaywalking, speeding tickets, subway stops
- Emirates Airlines, Delta Airlines, American Airlines using facial recognition technology for boarding
- India, Australia looking to follow China’s footsteps to build facial recognition database

Responsible CS (1/2)

Over 65% of adults across 26 countries support the government’s use of facial recognition, with regulations.

San Francisco, Portland, Oakland, etc. have issued bans on facial recognition use by the police and other government agencies.

Critics argue that crafting regulations is a more effective solution than outright bans.

Some just want to know how/where their faces are being tracked.

If deployed correctly, facial recognition can be valuable and potentially life-saving.

How much facial recognition can you tolerate? What are some things we can do to strike a balance between security/privacy?

Responsible CS (2/2)

Consequences:

- Over 65% of adults across 26 countries support the government’s use of facial recognition, with regulations.
- San Francisco, Portland, Oakland, etc. have issued bans on facial recognition use by the police and other government agencies.
- Critics argue that crafting regulations is a more effective solution than outright bans.
- Some just want to know how/where their faces are being tracked.
- If deployed correctly, facial recognition can be valuable and potentially life-saving.

Initial Survey (1/5)

- In which programming language do you feel comfortable programming, if any?
Lecture 10
Graphics Part III – Building up to Cartoon

Outline

- Shapes
  - example: MovingShape, App, PaneOrganizer, and MoveHandler classes
- Constants
  - Clicker Question: Slide 44
- Composite Shapes
  - example: Alien
  - Clicker Question: Slide 56, Slide 64
- Cartoon
Example: MovingShape

- Specification: App that displays a shape and buttons that shift position of the shape left and right by a fixed increment
- Purpose: Practice working with absolute positioning of Panes, various Shapes, and more event handling!

Process: MovingShapeApp

1. Write a top-level App class that extends javafx.application.Application and implements start (standard pattern)
2. Write a PaneOrganizer class that instantiates root node and makes a public getRoot() method. In PaneOrganizer, create an Ellipse and add it as child of root Pane
3. Write setupShape() and setupButtons() helper methods to be called within PaneOrganizer's constructor. These will factor out the code for creating our custom Pane
4. Register Buttons with EventHandlers that handle Button's ActionEvent (clicks) by moving Shape correspondingly

MovingShapeApp: App Class (1/3)

"NOTE: Exactly the same process as previous examples"

1a. Instantiate a PaneOrganizer and store it in the local variable organizer

```java
public class App extends Application {
    @Override
    public void start(Stage stage) {
        PaneOrganizer organizer = new PaneOrganizer();
        Scene scene = new Scene(organizer.getRoot(), 200, 200);
        stage.setScene(scene);
        stage.setTitle("Color Changer");
        stage.show();
    }
}
```
MovingShapeApp: App Class (2/3)

1a. Instantiate a PaneOrganizer and store it in the local variable optimizer.

```
public class App extends Application {

    public void start(Stage stage) {
        PaneOrganizer optimizer = new PaneOrganizer();
        Scene scene = new Scene(optimizer.getRoot(), 200, 200);
        stage.setScene(scene);
        stage.setTitle("Color Changer");
        stage.show();
    }
}
```

1b. Instantiate a Scene, passing in organizer.getRoot() and desired width and height of Scene (in this case 200x200).

```
Scene scene = new Scene(organizer.getRoot(), 200, 200);
```

*NOTE: Exactly the same process as previous examples*

MovingShapeApp: App Class (3/3)

1a. Instantiate a PaneOrganizer and store it in the local variable organizer.

```
public class App extends Application {

    public void start(Stage stage) {
        PaneOrganizer optimizer = new PaneOrganizer();
        Scene scene = new Scene(optimizer.getRoot(), 200, 200);
        stage.setScene(scene);
        stage.setTitle("Moving Shape!");
        stage.show();
    }
}
```

1b. Instantiate a Scene, passing in organizer.getRoot() and desired width and height of Scene (in this case 200x200).

```
Scene scene = new Scene(organizer.getRoot(), 200, 200);
```

1c. Set scene, set Stage's title and show it!

```
stage.setScene(scene);
stage.setTitle("Moving Shape!");
stage.show();
```

*NOTE: Exactly the same process as previous examples*

Process: MovingShapeApp

1. Write a top-level App class that extends javafx.application.Application and implements start (standard pattern)

2. Write a PaneOrganizer class that instantiates root node and makes a public getRoot() method. In PaneOrganizer, create an Ellipse and add it as child of root Pane

3. Write setupShape() and setupButtons() helper methods to be called within PaneOrganizer's constructor. These will factor out the code for creating our custom Pane

4. Register Buttons with EventHandlers that handle Buttons' ActionEvents (clicks) by moving 'Shape correspondingly'
MovingShapeApp: PaneOrganizer
Class (1/4)
2a. Instantiate the root Pane and store it in the instance variable _root

```java
public class PaneOrganizer {
    private Pane _root;
    private Ellipse _ellipse;
    public PaneOrganizer() {
        _root = new Pane();
    }
    ```

2b. Create a public `getRoot()` method that returns _root

```java
public Pane getRoot() {
    return _root;
}
```
MovingShapeApp: PaneOrganizer
Class (4/4)
2a. Instantiate the root Pane and store it in the instance variable _root
2b. Create a publicgetRoot() method that returns _root
2c. Instantiate the Ellipse and add it as a child of the root Pane
2d. Call setupShape() and setupButtons(), defined next

```
public class PaneOrganizer {
    private Pane _root;
    private Ellipse _ellipse;
    public PaneOrganizer() {
        _root = new Pane();
        _ellipse = new Ellipse(50, 50);
        _root.getChildren().add(_ellipse);
        this.setupShape();
        this.setupButtons();
    }
    public Pane getRoot() { return _root; }
}
```

Process: MovingShapeApp
1. Write a top-level App class that extends javafx.application.Application and implements start (standard pattern)
2. Write a PaneOrganizer class that instantiates root node and makes a publicgetRoot() method. In PaneOrganizer, create an Ellipse and add it as child of root Pane
3. Write setupShape() and setupButtons() helper methods to be called within PaneOrganizer’s constructor. These will factor out code for creating our custom Pane
4. Register Buttons with EventHandlers that handle Button’s ActionEvents (clicks) by moving Shape correspondingly

Aside: helper methods
- As our applications start getting more complex, we will need to write a lot more code to get the UI looking the way we would like
- Such code would convolute the PaneOrganizer constructor—it is good practice to factor out code into helper methods that are called within the constructor—another use of the delegation pattern
  - setupShape() fills and positions Ellipse
  - setupButtons() adds and positions Buttons, and registers them with their appropriate EventHandlers
- Helper methods of the form setupX() are fancy initializing assignments. Should be used to initialize variables, but not for arbitrary/non-initializing code.
- Generally, helper methods should be private – more on this in a moment
MovingShapeApp: setupShape() helper method

- For this application, "helper method" setupShape() will only set fill color and position Ellipse in Pane using absolute positioning.
- Helper method is private—why is this good practice?
  - Only PaneOrganizer() should be allowed to initialize the color and location of the Ellipse.
  - Private methods are not directly inherited and are not accessible to subclasses—though inherited superclass methods may make use of them if the subclass knows about them!

```java
public class PaneOrganizer {
  private Pane _root;
  private Ellipse _ellipse;

  public PaneOrganizer() {
    _root = new Pane();
    _ellipse = new Ellipse(50, 50);
    _root.getChildren().add(_ellipse);
    this.setupShape();
    this.setupButtons();
  }

  public Pane getRoot() {
    return _root;
  }

  private void setupShape() {
    _ellipse.setFill(Color.RED);
    _ellipse.setCenterX(50);
    _ellipse.setCenterY(50);
  }
}
```

Aside: PaneOrganizer Class (1/3)

- We were able to absolutely position _ellipse in the root Pane because our root is simply a Pane and not one of the more specialized subclasses.
- We could also use absolute positioning to position the Buttons in the Pane in our setUpButtons() method... But look how annoying trial-and-error is!

```java
left.relocate(50, 165);
right.relocate(120, 165);
left.relocate(100, 180);
right.relocate(150, 180);
left.relocate(50, 150);
right.relocate(120, 150);
left.relocate(50, 165);
right.relocate(120, 165);
```

Is there a better way? ...hint: leverage Scene Graph hierarchy and delegation!

Aside: PaneOrganizer Class (2/3)

- Rather than absolutely positioning Buttons directly in root Pane, use a specialized layout Pane: add a new HBox as a child of the root Pane.
- Add Buttons to HBox, to align horizontally.
- Continuing to improve our design, use a BorderPane as root to use its layout manager.
- Now need to add Ellipse to the root.
  - Could simply add Ellipse to CENTER of root BorderPane.
  - But this won't work—if BorderPane dictates placement of Ellipse we won't be able to update its position with Buttons.
- Instead, create a Pane to contain Ellipse and add the Pane as child of root. Can adjust Ellipse within its shapePane independently!
Aside: PaneOrganizer Class (3/3)

- This makes use of the built-in layout capabilities available to us in JavaFX!
- Also makes symmetry between the panel holding a shape (in Cartoon, this panel will hold composite shapes that you'll make) and the panel holding our buttons
- Note: this is only one of many design choices for this application!
  - keep in mind all of the different layout options when designing your programs!
  - using absolute positioning for entire program is most likely not best solution—where possible, leverage power of layout managers (BorderPane, HBox, VBox,...)

MovingShapeApp: update to BorderPane

3a. Change root to a BorderPane, create a Pane to contain Ellipse

```java
public class PaneOrganizer {
    private BorderPane _root;
    private Ellipse _ellipse;

    public PaneOrganizer() {
        _root = new BorderPane();
        // setup shape pane
        Pane shapePane = new Pane();
        _ellipse = new Ellipse(50, 50);
        shapePane.getChildren().add(_ellipse);
        _root.setCenter(shapePane);
        this.setupShape();
        this.setupButtons();
    }

    private void setupButtons() {
        // more code to come!
    }
}
```

3b. To add shapePane to center of BorderPane, call setCenter(shapePane) on root

- note: none of the code in our setShape() method needs to be updated since it accesses _ellipse directly... with this redesign, _ellipse now is just graphically contained within a different Pane (the shapePane) and now in the center of the root because we called setCenter(shapePane)
- and PaneOrganizer can still access the ellipse because it remains its instance variable!
  - this could be useful if we want to change any properties of the Ellipse later on, e.g., updating its x and y position, or changing its color
- illustration of graphical vs. logical containment
MovingShapeApp: setupButtons() method (1/5)
3c. Instantiate a new HBox, then add it as a child of BorderPane, in bottom position

MovingShapeApp: setupButtons() method (2/5)
3d. Instantiate two Buttons

MovingShapeApp: setupButtons() method (3/5)
3e. Add the Buttons as children of the new HBox
MovingShapeApp: setupButtons() method (4/5)

3f. Set horizontal spacing between Buttons as you like

public class PaneOrganizer {
    private BorderPane _root;
    private Ellipse _ellipse;
    public PaneOrganizer() {
        _root = new BorderPane();
        // setup of shape pane and shape elided!
        this.setupButtons();
    }
    private void setupButtons() {
        HBox buttonPane = new HBox();
        _root.setBottom(buttonPane);
        Button b1 = new Button("Move Left");
        Button b2 = new Button("Move Right");
        buttonPane.getChildren().addAll(b1, b2);
        buttonPane.setSpacing(30);
    }
}

MovingShapeApp: setupButtons() method (5/5)

3g. We will come back to the PaneOrganizer class in the next step in order to register Buttons with their EventHandlers, but first we should define the EventHandler

public class PaneOrganizer {
    private BorderPane _root;
    private Ellipse _ellipse;
    public PaneOrganizer() {
        _root = new BorderPane();
        // setup of shape pane and shape elided!
        this.setupButtons();
    }
    private void setupButtons() {
        HBox buttonPane = new HBox();
        _root.setBottom(buttonPane);
        Button b1 = new Button("Move Left");
        Button b2 = new Button("Move Right");
        buttonPane.getChildren().addAll(b1, b2);
        buttonPane.setSpacing(30);
    }
}

Process: MovingShapeApp

1. Write a top-level App class that extends javafx.application.Application and implements start (standard pattern)
2. Write a PaneOrganizer class that instantiates root node and makes a public getRoot() method. In PaneOrganizer, create an Ellipse and add it as child of root Pane
3. Write setupShape() and setupButtons() helper methods to be called within PaneOrganizer’s constructor. These will factor out the code for creating our custom Pane
4. Register Buttons with inner class EventHandlers that handle Buttons’ ActionEvents (clicks) by moving Shape correspondingly
Aside: Creating EventHandlers

- Our goal is to register each button with an EventHandler
  - the "Move Left" Button moves the Ellipse left by a set amount
  - the "Move Right" Button moves the Ellipse right the same amount
- We could define two separate EventHandlers, one for the "Move Left" Button and one for the "Move Right" Button...
  - why might this not be the optimal design?
  - remember, we want to be efficient with our code usage!
- Instead, we can define one EventHandler
  - factor out common behavior into one class that will have two instances
  - specifics determined by parameters passed into the constructor!
  - admittedly, this is not an obvious design—these kinds of simplifications typically have to be learned...

34

MovingShapeApp: MoveHandler (1/3)

4a. Declare an instance variable `_distance` that will be initialized differently depending on whether the `isLeft` argument is true or false

```java
public class PaneOrganizer {
    // other code elided
    public PaneOrganizer() {
        // other code elided
    }
    // private class MoveHandler implements EventHandler<ActionEvent>
    private class MoveHandler implements EventHandler<ActionEvent> {
        private int _distance;
        public MoveHandler(boolean isLeft) {
            // other code elided
        }
        public void handle(ActionEvent e) {
            // other code elided
        }
    }
    // other code elided
}
```

35

MovingShapeApp: MoveHandler (2/3)

4b. Set `_distance` to 10 initially—if the registered Button `isLeft` change `_distance` to -10 so the Ellipse moves in the opposite direction

```java
public class PaneOrganizer {
    // other code elided
    public PaneOrganizer() {
        // other code elided
    }
    // private class MoveHandler implements EventHandler<ActionEvent>
    private class MoveHandler implements EventHandler<ActionEvent> {
        private int _distance;
        public MoveHandler(boolean isLeft) {
            _distance = 10;
            // other code elided
        }
        public void handle(ActionEvent e) {
            // other code elided
        }
    }
    // other code elided
}
```

36
MovingShapeApp: MoveHandler (3/3)

4a. Declare an instance variable `distance` that will be initialized differently depending on whether the `isLeft` argument is true or false

4b. Set `distance` to 10 initially – if the registered button `isLeft`, change `distance` to -10 so the ellipse moves in the opposite direction

4c. Implement the handle method to move the ellipse by `distance` in the horizontal direction

MovingShapeApp: back to setupButtons()

Register Buttons with their EventHandlers by calling `setOnAction()` and passing in our instances of MoveHandler, which we just created!

The Whole App
Reminder: Constants Class

- In our MovingShapeApp, we've been using absolute numbers in various places, not very extensible! What if we wanted to quickly change the size of our Scene or Shape to improve compile time?
- Our Constants class will keep track of a few important numbers.

```java
public class Constants {
    // units all in pixels
    public static final double X_RAD = 50;
    public static final double Y_RAD = 50;
    public static final double APP_WIDTH = 200;
    public static final double APP_HEIGHT = 130;
    public static final double BUTTON_SPACING = 30;
    // X_OFFSET is the graphical offset from the edge of the screen to where we want the X value of the Ellipse
    public static final double X_OFFSET = 100;
    public static final double Y_OFFSET = 50;
    public static final double DISTANCE_X = 10;
}
```

Clicker Question

When should you define a value in a Constants class?

A. When you use the value in more than one place.
B. Whenever the value will not change throughout the course of the program.
C. When the value is nontrivial (i.e., not 0 or 1)
D. All of the above.

The Whole App

```java
public class App extends Application {
    @Override
    public void start(Stage stage) {
        PaneOrganizer organizer = new PaneOrganizer();
        Scene scene = new Scene(organizer.getRoot(), Constants.APP_WIDTH, Constants.APP_HEIGHT);
        stage.setScene(scene);
        stage.setTitle("MovingShape!");
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```
Creating Composite Shapes

• What if we want to display something more elaborate than a single, simple geometric primitive?
• We can make a composite shape by combining two or more shapes!

Specifications: MovingAlien

• Transform MovingShape into MovingAlien
• An alien should be displayed on the central Pane, and should be moved back and forth by Buttons

MovingAlien: Design

• Create a class, Alien, to model a composite shape
• Define composite shape’s capabilities in Alien class
• Give Alien a setLocation() method that positions each component (face, left eye, right eye, all Ellipses)
  ○ another example of delegation pattern
Process: Turning MovingShape into MovingAlien

1. Create Alien class to model composite shape, and add each component of Alien to alienPane's list of children
2. Be sure to explicitly define any methods that we need to call on Alien from within PaneOrganizer, such as location setter/getter methods!
3. Modify PaneOrganizer to contain an Alien instead of an Ellipse

Alien Class

- The Alien class is our composite shape
- It contains three Ellipses—one for the face and one for each eye
- Constructor instantiates these Ellipses, sets their initial sizes/colors, and adds them as children of the alienPane—which was passed in as a parameter
- Although Alien class deals with each component of the composite shape individually, every component should reside on the same pane as all other components
  - Thus, must pass pane as a parameter to allow methods for manipulating composite shape in pane

```java
public class Alien {
    private Ellipse _face;
    private Ellipse _leftEye;
    private Ellipse _rightEye;

    public Alien(               ) {//Alien lives in passed Pane
        _face = new Ellipse(Constants.X_RAD, Constants.Y_RAD);
        _face.setFill(Color.CHARTREUSE);
        /*EYE_X and EYE_Y are constants referring to the width and height of the eyes, the eyes' location/center is changed later
        */
        _leftEye = new Ellipse(Constants.EYE_X, Constants.EYE_Y);
        _leftEye.setFill(Color.BLACK);
        _rightEye = new Ellipse(Constants.EYE_X, Constants.EYE_Y);
        _rightEye.setFill(Color.BLACK);
        alienPane.getChildren().addAll(_face, _leftEye, _rightEye);
        this.setXLoc(Constants.START_X_OFFSET);
    }

    public void setXLoc(double x) {
        _face.setCenterX(x - Constants.EY
```
Alien Class

• In MovingShapeApp, the following call is made from within our MoveHandler's handle method in order to move the ellipse:
  ```java
  ellipse.setCenterX(ellipse.getCenterX() + _distance);
  ```

• Because we called JavaFX's getCenterX() and setCenterX(_) on our shape from within the PaneOrganizer class, we must now define our own equivalent methods such as setLocX(_) and getLocX() to set the Alien's location in the Alien class!

• This allows our Alien class to function like an ellipse in our program!

• Note: most of the time when you are creating complex shapes, you will want to define a more extensive setLocation(double x, double y) method rather than having a separate method for the X or Y location.

```
public class Alien {
  private Ellipse _face;
  private Ellipse _leftEye;
  private Ellipse _rightEye;

  public Alien(Pane alienPane) {
    _face = new Ellipse(Constants.X_RAD, Constants.Y_RAD);
    _face.setFill(Color.CHARTREUSE);
    _leftEye = new Ellipse(Constants.EYE_X, Constants.EYE_Y);
    _leftEye.setFill(Color.BLACK);
    _rightEye = new Ellipse(Constants.EYE_X, Constants.EYE_Y);
    _rightEye.setFill(Color.BLACK);
    alienPane.getChildren().addAll(_face, _leftEye, _rightEye);
  }

  public void setXLoc(double x) {
    _face.setCenterX(x);
    _leftEye.setCenterX(x - Constants.EYE_OFFSET);
    _rightEye.setCenterX(x + Constants.EYE_OFFSET);
  }

  public double getXLoc() {
    return _face.getCenterX();
  }
```

MovingAlien: Alien Class (1/3)

2a. Define Alien's setXLoc(...) by setting center X of face, left and right eyes (same for setYLoc); note use of additional constants
   • note: relative positions between the Ellipses remains the same

MovingAlien: Alien Class (2/3)

2a. Define Alien's setXLoc(...) by setting center X of face, left and right eyes (same for setYLoc);
   • note: relative positions between the Ellipses remains the same

2b. Define getXLoc() method:
   the horizontal center of the Alien will always be center of the face Ellipse
MovingAlien: Alien Class (3/3)

2a. Define Alien's setXLoc() by setting center X of face, left and right eyes (same for setYLoc);
   note: relative positions between the Ellipses remains the same
2b. Define getXLoc() method: the horizontal center of the Alien will always be center of _Face Ellipse
2c. Set starting X location of Alien in constructor!

```java
public class Alien {
    private Ellipse _face;
    private Ellipse _leftEye;
    private Ellipse _rightEye;

    public Alien(Pane alienPane) {
        _face = new Ellipse(Constants.X_RAD, Constants.Y_RAD);
        _face.setFill(Color.CHARTREUSE);
        _leftEye = new Ellipse(Constants.EYE_X, Constants.EYE_Y);
        _leftEye.setFill(Color.BLACK);
        _rightEye = new Ellipse(Constants.EYE_X, Constants.EYE_Y);
        _rightEye.setFill(Color.BLACK);
        alienPane.getChildren().addAll(_face, _leftEye, _rightEye);
        this.setXLoc(Constants.START_X_OFFSET);
    }

    public void setXLoc(double x) {
        _face.setCenterX(x);
        _leftEye.setCenterX(x - Constants.EYE_OFFSET);
        _rightEye.setCenterX(x + Constants.EYE_OFFSET);
    }

    public double getXLoc() {
        return _face.getCenterX();
    }
}
```

Clicker Question

Which House constructor makes the correct composite shape, given the rest of the program is set up correctly?

- A.
- B.
- C.
- D.

Process: Turning MovingShape into MovingAlien

1. Create Alien class to model composite shape, and add each component of Alien to alienPane's list of children
2. Be sure to explicitly define any methods that we need to call on Alien from within PaneOrganizer, such as location setter/getter methods!
3. Modify PaneOrganizer to contain an Alien instead of an Ellipse
MovingAlien: PaneOrganizer
Class (1/4)
• Only have to make a few changes to PaneOrganizer!
• Instead of knowing about an Ellipse called _ellipse, knows about an Alien called _alien
• Change the shapePane to be an alienPane (we could have called it anything!)

```java
public class PaneOrganizer {
    private BorderPane _root;
    private Alien _alien;
    public PaneOrganizer () {
        _root = new BorderPane();
        Pane alienPane = new Pane();
        _alien = new Alien(alienPane);
        _root.setCenter(alienPane);
        this.setupShape();
        this.setupButtons();
    }
    public Pane getRoot () {
        return _root;
    }
    private void setupShape () {
        _ellipse.setFill(Color.RED);
        _ellipse.setCenterX(Constants.X_OFFSET);
        _ellipse.setCenterY(Constants.Y_OFFSET);
    }
    private void setupButtons () {
        HBox buttonPane = new HBox();
        _root.setBottom(buttonPane);
        Button b1 = new Button("Move Left");
        Button b2 = new Button("Move Right");
        buttonPane.getChildren().addAll(b1, b2);
        buttonPane.setSpacing(30);
        b1.setOnAction(new MoveHandler(true));
        b2.setOnAction(new MoveHandler(false));
    }
} /* private class MoveHandler elided */
```

MovingAlien: PaneOrganizer
Class (2/4)
• setupShape() method is no longer needed, as we now setup the Alien within the Alien class

```java
public class PaneOrganizer {
    private BorderPane _root;
    private Alien _alien;
    public PaneOrganizer () {
        _root = new BorderPane();
        Pane alienPane = new Pane();
        _alien = new Alien(alienPane);
        _root.setCenter(alienPane);
        //this.setupShape();
        this.setupButtons();
    }
    public Pane getRoot () {
        return _root;
    }
    //private void setupShape () {
    //    _ellipse.setFill(Color.RED);
    //    _ellipse.setCenterX(Constants.X_OFFSET);
    //    _ellipse.setCenterY(Constants.Y_OFFSET);
    //}
    private void setupButtons () {
        HBox buttonPane = new HBox();
        _root.setBottom(buttonPane);
        Button b1 = new Button("Move Left");
        Button b2 = new Button("Move Right");
        buttonPane.getChildren().addAll(b1, b2);
        buttonPane.setSpacing(30);
        b1.setOnAction(new MoveHandler(true));
        b2.setOnAction(new MoveHandler(false));
    }
} /* private class MoveHandler elided */
```

MovingAlien: PaneOrganizer
Class (3/4)
• setupShape() method is no longer needed, as we now setup the Alien within the Alien class
• remember that we set a default location for the Alien in its constructor.

```java
public class PaneOrganizer {
    private BorderPane _root;
    private Alien _alien;
    public PaneOrganizer () {
        _root = new BorderPane();
        Pane alienPane = new Pane();
        _alien = new Alien(alienPane);
        _root.setCenter(alienPane);
        //this.setupShape();
        //    _ellipse.setFill(Color.RED);
        //    _ellipse.setCenterX(Constants.X_OFFSET);
        //    _ellipse.setCenterY(Constants.Y_OFFSET);
    }
    public Pane getRoot () {
        return _root;
    }
    //private void setupShape () {
    //    _ellipse.setFill(Color.RED);
    //    _ellipse.setCenterX(Constants.X_OFFSET);
    //    _ellipse.setCenterY(Constants.Y_OFFSET);
    //}
    private void setupButtons () {
        HBox buttonPane = new HBox();
        _root.setBottom(buttonPane);
        Button b1 = new Button("Move Left");
        Button b2 = new Button("Move Right");
        buttonPane.getChildren().addAll(b1, b2);
        buttonPane.setSpacing(30);
        b1.setOnAction(new MoveHandler(true));
        b2.setOnAction(new MoveHandler(false));
    }
} /* private class MoveHandler elided */
```
MovingAlien: PaneOrganizer
Class (4/4)

- Last modification we have to make is from within the MoveHandler class, where we will swap in _alien for _ellipse references
- We implemented setXLoc(…) and getXLoc(…) methods in Alien so MoveHandler can call them

The Whole App

Additional Classes

- Notice how we created another class for our Alien composite shape instead of simply adding each individual shape to PaneOrganizer
- As your programs get more complex (e.g., two shapes interacting with one another, shapes changing color, etc.), you may want to create even more additional classes that perform the desired functions instead of doing everything in PaneOrganizer
  - for example, if we are trying to create a Tic Tac Toe app, all of the game logic should go into a separate class; PaneOrganizer would only be responsible for placing Pane and other elements on the screen
  - this will make PaneOrganizer less cluttered and your program as a whole much easier to read
  - keep this in mind for your upcoming assignments!!
Clicker Question

What is the best practice for setting up graphical scenes (according to CS15)?

A. Absolutely position everything using trial and error, and use as few panes as possible.
B. Have any shape be contained in its own pane, and only make classes for composite shapes of more than 5 shapes.
C. Use a top-level class, make classes for more complicated shapes, and store composite shapes, or just generally related objects, within panes.

Your Next Project: Cartoon! (1/2)

• You’ll be building a JavaFX application that displays your own custom “cartoon”, much like the examples in this lecture
• But your cartoon will be animated!

Your Next Project: Cartoon! (2/2)

• How can we animate our cartoon (e.g. make the cartoon move across the screen)?
• As in film and video animation, can create apparent motion with many small changes in position
• If we move fast enough and in small enough increments, we get smooth motion!
• Same goes for smoothly changing size, orientation, shape, etc.
Animation in Cartoon

- Use a **TimeLine** to create incremental change
- It’ll be up to you to figure out the details… but for each repetition of the **KeyFrame**, your cartoon should move (or change in other ways) a small amount!
  - reminder: if we move fast enough and in small enough increments, we get smooth motion!

---

Announcements

- Cartoon has been released!
  - Early Handin: Tuesday, 10/15 at 11:59pm
  - On-Time Handin: Thursday, 10/17 at 11:59pm
  - Late Handin: Saturday, 10/19 at 11:59pm
- Top 6 Cartoons win a meal with Andy
- Cartoon **help session** this Thursday 8:30 – 10 PM at CIT 165 (Motorola)!!
- Section has 2 parts this week: Cartoon check-in and lab
  - Meet at normal section time at the Sunlab to get practice with JavaFX
  - Section TAs will send out signups for you to go over your design for Cartoon, and get to connect with your section TAs