WiCS
Women In Computer Science

Join our listserv to get updates and hear opportunities!
Email wics@lists.cs.brown.edu

To apply to be a mentee in the WiCS mentorship program go to http://tinyurl.com/h9jpwmv
Outline

• Shapes
  • Example: MovingShape
  • App, PaneOrganizer, and MoveHandler classes

• Constants

• Composite Shapes
  • Example: Alien

• Cartoon
Example: MovingShape

- Spec: App that displays a shape and buttons that shift position of the shape left and right by a fixed increment
- Practice working with absolute positioning of Panes, various Shapes, & 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 Buttons’ ActionEvents (clicks) by moving Shape correspondingly**
(1) Top-level Class: MovingShapeApp

*NOTE: Exactly the same process as previous examples*

A. 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();
    }
}
```
(1) **Top-level Class**: `MovingShapeApp`  

*NOTE: Exactly the same process as previous examples*

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

B. Instantiate a `Scene`, passing in `organizer.getRoot()` and desired width and height of `Scene`

```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();
    }
}
```
(1) Top-level Class: **MovingShapeApp**

*NOTE: Exactly the same process as previous examples*

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

B. Instantiate a `Scene`, passing in `organizer.getRoot()` and desired width and height of `Scene`

C. Set `scene`, set Stage’s title and show it!

```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("Moving Shape!");
        stage.show();
    }
}
```
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
A. Instantiate the root Pane and store it in the instance variable _root

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

    public PaneOrganizer() {
        _root = new Pane();
    }
}
```
(2) PaneOrganizer Class (2/4)

A. Instantiate the root Pane and store it in the instance variable _root

B. Create a public getRoot() method that returns _root

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

    public PaneOrganizer() {
        _root = new Pane();
    }

    public Pane getRoot() {
        return _root;
    }
}
```
A. Instantiate the root `Pane` and store it in the instance variable `_root`.

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

C. Instantiate the `Ellipse` and add it as child of the root `Pane`.

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

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

    public Pane getRoot() {
        return _root;
    }
}
```
(2) **PaneOrganizer** Class (4/4)

A. Instantiate the root **Pane** and store it in the instance variable `_root`

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

C. Instantiate the **Ellipse** and the **Buttons**, and add them as children of the root **Pane**

D. **Call `setupShape()` and `setupButtons()`**, defined next

```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;
    }
}
```
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
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.

- `setupShape()` fills and positions Ellipse
- `setupButtons()` adds and positions Buttons, and registers them with their appropriate EventHandlers
**(3) PaneOrganizer: 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 the `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 methods may make use of them!
  - generally, helper methods should be `private`

```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);
    }
}
```
Digression: PaneOrganizer Class (1/3)

• We were able to absolutely position (position is fixed, cannot be changed) _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!

Is there a better way? ...hint: leverage Scene Graph hierarchy and delegation!
**Digression: 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
- Now need to add need **Ellipse** to the root
  - could simply add **Ellipse** to center of root
  - but this won’t work—if we let the **BorderPane** dictate the placement of the **Ellipse** we won’t be able to update the position of it using the **Buttons**
  - instead: create a **Pane** to contain **Ellipse** and add the **Pane** as child of root!
**Digression: PaneOrganizer Class (3/3)**

- This makes use of the built-in layout capabilities available to us in JavaFX!
- Note: this is only one of *many* design choices for this application!
  - keep in mind all of your different layout options when designing your programs!
  - absolutely positioning the entire program is most likely *not* the best solution
(3) PaneOrganizer  
Class: update to BorderPane

A. 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);

        this.setupShape();
        this.setupButtons();
    }

    private void setupButtons(){
        // more code to come!
    }
}
```
(3) PaneOrganizer Class: update to BorderPane

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

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

• note: none of the code in our setupShape() method needs to be updated – our Ellipse is just graphically contained within a different Pane, the shapePane

• but PaneOrganizer still can access it!
  • 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

```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!
    }
}
```
(3) PaneOrganizer Class: setupButtons() method

C. Instantiate a new HBox, then add it as a child of BorderPane, in bottom position

```java
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);
    }
}
```
(3) PaneOrganizer Class: setupButtons() method

C. Instantiate a new HBox, then add it as a child of BorderPane, in bottom position

D. Instantiate two Buttons

```java
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”);
    }
}
```
(3) PaneOrganizer Class: setupButtons() method

C. Instantiate a new HBox, then add it as a child of BorderPane, in bottom position

D. Instantiate two Buttons

E. Add the Buttons as children of the new HBox
   • note: different from before — now adding Buttons as children of HBox
   • remember that a Node, like a Button, can have at most one parent!

```java
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);
    }
}
```
(3) PaneOrganizer Class: setupButtons() method

F. Set horizontal spacing between Buttons as you like

```java
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);
    }
}
```
(3) PaneOrganizer Class: setupButtons() method

F. Set horizontal spacing between Buttons as you like

G. Register Buttons with their EventHandlers by calling setOnAction() and passing in our instances of MoveHandler, which we will create next!

```java
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);
        b1.setOnAction(new MoveHandler(true));
        b2.setOnAction(new MoveHandler(false));
    }
}
```
Process: MovingShapeApp

1. Write a `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
**Digression: 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
  - specifics determined by parameters passed into the constructor!
(4) **MoveHandler**: Private Inner Class (1/3)

A. 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 int _distance;

        public MoveHandler(boolean isLeft) {

        }

        public void handle(ActionEvent e) {
        }
    }
}
```
**A.** Declare an instance variable `_distance` that will be initialized differently depending on whether the `isLeft` argument is `true` or `false`.

**B.** 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 int _distance;

        public MoveHandler(boolean isLeft) {
            _distance = 10;
            if (isLeft) {
                _distance *= -1; // change sign
            }
        }

        public void handle(ActionEvent e) {
        }
    }
}
```
(4) **MoveHandler**: Private Inner Class (3/3)

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

B. Set `_distance` to 10 initially – if the registered `Button` `isLeft`, change `_distance` to -10 so the `Ellipse` moves in the opposite direction

C. Implement the handle method to move the `Ellipse` by `_distance` in the horizontal direction

```java
public class PaneOrganizer {
    // other code elided

    public PaneOrganizer() {
        // other code elided
    }

    private class MoveHandler implements EventHandler<ActionEvent> {
        private int _distance;

        public MoveHandler(boolean isLeft) {
            _distance = 10;
            if (isLeft) {
                _distance *= -1; // change sign
            }
        }

        public void handle(ActionEvent e) {
            _ellipse.setCenterX(_ellipse.getCenterX() + _distance);
        }
    }
}
```
The Whole App

package MovingShape;

// imports for the App class
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;

// imports for the PaneOrganizer class
import javafx.event.*;
import javafx.geometry.Pos;
import javafx.scene.control.Button;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
import javafx.scene.shape.Ellipse;

public class App extends Application {
    @Override
    public void start(Stage stage) {
        PaneOrganizer organizer = new PaneOrganizer();
        Scene scene = new Scene(organizer.getRoot(), 200, 130);
        stage.setScene(scene);
        stage.setTitle("MovingShape!");
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}

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

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

    public Pane getRoot() {
        return _root;
    }

    private void setupShape() {
        _ellipse.setFill(Color.RED);
        _ellipse.setCenterX(100);
        _ellipse.setCenterY(50);
    }

    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(left, right);
        buttonPane.setSpacing(30);
        buttonPane.setAlignment(Pos.CENTER);
        b1.setOnAction(new MoveHandler(true));
        b2.setOnAction(new MoveHandler(false));
    }

    private class MoveHandler implements EventHandler<ActionEvent> {
        private int _distance;
        public MoveHandler(boolean isLeft) {
            _distance = 10;
            if (isLeft) {
                _distance *= -1;
            }
        }

        public void handle(ActionEvent event) {
            _ellipse.setCenterX(_ellipse.getCenterX() + _distance);
        }
    }
}

public class BorderPane {
    public BorderPane() {
        Pane shapePane = new Pane();
        this.shapePane.setCenterX(100);
        this.shapePane.setCenterY(50);
    }
}

public class Ellipse {
    public Ellipse(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public void draw() {
        GraphicsContext context = new GraphicsContext();
        context.setFill(Color.RED);
        context.fillOval(x, y, 50, 50);
    }
}

public class MoveHandler {
    public MoveHandler(bool isLeft) {
        _distance = 10;
        if (isLeft) {
            _distance *= -1;
        }
    }

    public void handle(ActionEvent event) {
        _ellipse.setCenterX(_ellipse.getCenterX() + _distance);
    }
}
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?
• Our Constants class will keep track of a few important numbers
• For our MovingShapeApp, make constants for width and height of the Ellipse and of the Panel it sits in, as well as the start location and move distance

```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;
    /* ELLIPSE_CENTER_X is the graphical offset from the edge of the screen to where we want the X value of the Ellipse */
    public static final double ELLIPSE_CENTER_X = 100;
    public static final double ELLIPSE_CENTER_Y = 50;
    public static final double DISTANCE_X = 10;
}
```
Clicker Question

When should you define a number in a Constants class?

A. When you use the number in more than one place.
B. Whenever the number's value will not change throughout the course of the program.
C. When the number's name is really long.
D. All of the above.
The Whole App

no more literal numbers = much better design!

Constants

class elided

_distance = 10;

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!
Spec: 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**)  
  • example of **delegation**
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

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

    public Alien(Pane alienPane) {//Alien lives in passed Pane
        _face = new Ellipse(Constants.X_RAD, Constants.Y_RAD);
        _face.setFill(Color.CHARTREUSE);
        /*EYE_W and EYE_H are constants referring to the width and height of the eyes, the eyes' location/center is changed later in the program.*/
        _leftEye = new Ellipse(Constants.EYE_W, Constants.EYE_H);
        _leftEye.setFill(Color.BLACK);
        _rightEye = new Ellipse(Constants.EYE_W, Constants.EYE_H);
        _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();
    }
}
```
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

• In MovingShapeApp, the following call is made from within our MoveHandler’s handle method in order to move the Ellipse:

  _ellipse.setCenterX(_ellipse.getCenterX() + _distance);

• Because we call Java-FX’s getCenterX() and setCenterX(...) on our shape from within the PaneOrganizer class, we must 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
Alien Class (1/3)

A. 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

```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_W, Constants.EYE_H);
        _leftEye.setFill(Color.BLACK);
        _rightEye = new Ellipse(Constants.EYE_W, Constants.EYE_H);
        _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);
    }
}
```
Alien Class (2/3)

A. Define Alien’s setXLoc(...) by setting center X of face, left and right eyes (same for setYLoc)
   • note: relative positions between the Ellipse’s remains the same

B. Define getXLoc() method: the horizontal center of the Alien will always be center of face Ellipse

```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_W, Constants.EYE_H);
        _leftEye.setFill(Color.BLACK);
        _rightEye = new Ellipse(Constants.EYE_W, Constants.EYE_H);
        _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();
    }
}
```
A. 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

B. Define getXLoc() method: the horizontal center of the Alien will always be center of face Ellipse

C. 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_W, Constants.EYE_H);
        _leftEye.setFill(Color.BLACK);
        _rightEye = new Ellipse(Constants.EYE_W, Constants.EYE_H);
        _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();
    }
}
```
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
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 */
}
```
• `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 */
}
```
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
this.setXLoc(Constants.START_X_OFFSET);
```
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
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("MovingAlien!");
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}

public class Alien {
    private Ellipse _face;
    private Ellipse _leftEye;
    private Ellipse _rightEye;
    public Alien(Pane root) {
        _face = new Ellipse(Constants.X_RAD, Constants.Y_RAD);
        _face.setFill(Color.CHARTREUSE);
        _leftEye = new Ellipse(Constants.EYE_W, Constants.EYE_H);
        _rightEye = new Ellipse(Constants.EYE_W, Constants.EYE_H);
        this.setXLoc(Constants.START_X_OFFSET);
        root.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();
    }
}

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.setupButtons();
    }

    public Pane getRoot() {
        return _root;
    }

    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 implements EventHandler<ActionEvent> {
        private int _distance;
        public MoveHandler(boolean isLeft) {
            _distance = Constants.DISTANCE_X;
            if (isLeft) {
                _distance *= -1;
            }
        }

        public void handle(ActionEvent event) {
            _alien.setXLoc(_alien.getXLoc() + _distance);
        }
    }
}
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 (ex. 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 TicTacToe
    • TicTacToe checks for game over, valid moves, etc
    • PaneOrganizer places Panes 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!
Announcements

• Cartoon is released today!
  • You must hand in the mini-assignment by Thursday at 2pm
  • You should have received an email from a TA setting up design checks. If you haven't please email the HTAs
• Even if you don't have a minimally functional project by the deadline, hand it in!
• Projects are starting to get bigger- start early, start today, start yesterday!
• Health and Wellness Town Hall Thursday 10/13, 6-7:30, B&H168