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 Button’s 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 _root
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!

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 method that returns 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 ActionEvents (clicks) by moving Shape correspondingly

(2) PaneOrganizer Class (1/4)
A. Instantiate the root Pane and store it in the local variable _root
B. Create a public method that returns _root
C. Instantiate the Ellipse and add it as child of the root Pane

(2) PaneOrganizer Class (2/4)

(2) PaneOrganizer Class (3/4)
Is there a better way? Hint: Leverage Scene Graph hierarchy and delegation!

(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

Process: MovingShapeApp

1. Write a top-level Java class that extends javafx.application.Application and implements start (standard pattern)
2. Write a PaneOrganizer class that instantiates a root node and makes a public getRoot() method. In PaneOrganizer, create an Ellipse and add it as a 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

(3) PaneOrganizer Class

• 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

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 pane is 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 and verbose it is!

Digression: PaneOrganizer Class (2/3)

• Rather than absolutely positioning Buttons directly in root Pane, use a specialized layoutPane: 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 Eclipse to the root
• But this won’t work! See how the BorderPane decides the placement of the Ellipse. We won’t be able to update the position off using the Buttons
• Instead, create a Pane to contain Ellipse and add the Pane as a child of root

4/17 10/11/16
**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

**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 taken, e.g., updating its x and y positions or changing its color

**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!
3. PaneOrganizer Class: setupButtons() method
   F. Set horizontal spacing between buttons as you like

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

   • 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
   B. Set distance to 10 initially — if the registered Button isLeft change distance to -10 so the Ellipse moves in the opposite direction

4. MoveHandler: Private Inner Class (2/3)
   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
(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 is `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

Constants Class

- In our MovingShapeApp, we’ve been using absolute numbers in various places
  - Not very flexible if we wanted to quickly change the size of our Scene or Shapes?
  - 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

The Whole App

no more literal numbers = much better design!

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.

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

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 calls are made from within our MoveHandler's handle method in order to move the Ellipse:
  ellipse.setCenter(ellipse.getCenter() + _distance);
- Because we call Java-FX's getCenter() and setCenter(...) on our shape from within the PaneOrganizer class, we must define our own equivalent methods such as setLocation() and getLoc() 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 Alien()
```

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 void setupShape();

```

**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 Ellipses remains the same

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

```java
public int getXLoc();
```

**Alien Class (3/3)**

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 void setupShape();
```

**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

**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 PaneOrganizer()
```

**PaneOrganizer Class (2/4)**

- `setupShape()` method is no longer needed, as we now setup the Alien within the Alien class

```java
public void setupShape();
```
**PaneOrganizer Class (3/4)**

- `setupShape()` method is no longer needed, as we now set the `Alien` within the `Alien` class.
- Remember that we set a default location for the `Alien` in its constructor.

**The Whole App**

- `setupScene()` method calls `setupShape()` for each shape.
- `setupRoot()` method sets the root pane and adds shapes.

**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.

**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`.
- For example, `PaneOrganizer` is 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