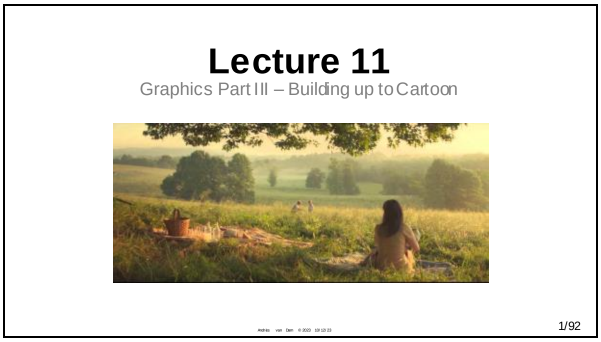
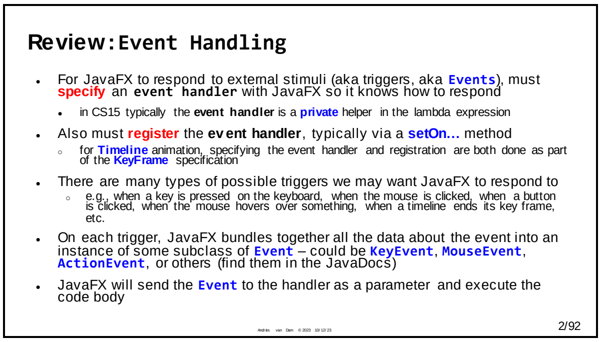


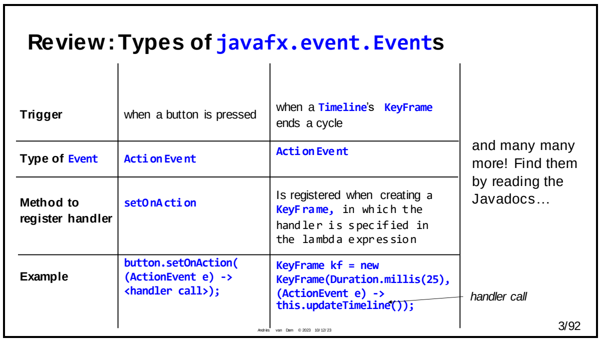
Lecture 11

Graphics Part III – Building up to Cartoon



Review: Event Handling

- For JavaFX to respond to external stimuli (aka triggers, aka **`Events`**), must **specify** an **`event handler`** with JavaFX so it knows how to respond
  - in CS15 typically the **event handler** is a **private** helper in the lambda expression
- Also must **register** the **event handler**, typically via a **setOn...** method
  - for **`Timeline`** animation, specifying the event handler and registration are both done as part of the **KeyFrame** specification
- There are many types of possible triggers we may want JavaFX to respond to
  - e.g., when a key is pressed on the keyboard, when the mouse is clicked, when a button is clicked, when the mouse hovers over something, when a timeline ends its key frame, etc.
- On each trigger, JavaFX bundles together all the data about the event into an instance of some subclass of **`Event`** – could be **`KeyEvent`**, **`MouseEvent`**, **`ActionEvent`**, or others (find them in the JavaDocs)
- JavaFX will send the **`Event`** to the handler as a parameter and execute the code body



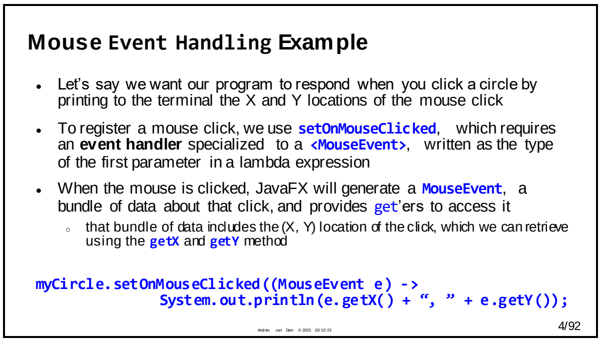
Review: Types of `javafx.event.Events`

Trigger	when a button is pressed	when a **`Timeline`**'s **`KeyFrame`** ends a cycle
Type of `Event`	**`ActionEvent`**	**`ActionEvent`**
Method to register handler	**`setOnAction`**	Is registered when creating a **`KeyFrame`**, in which the handler is specified in the lambda expression

Example	`button.setOnAction(`
	`(ActionEvent e) ->`
	`<handler call>);`
		`KeyFrame kf = new`
		`KeyFrame(Duration.millis(25),`
		`(ActionEvent e) ->`
		`this.updateTimeline());`

and many many more! Find them by reading the Javadocs…

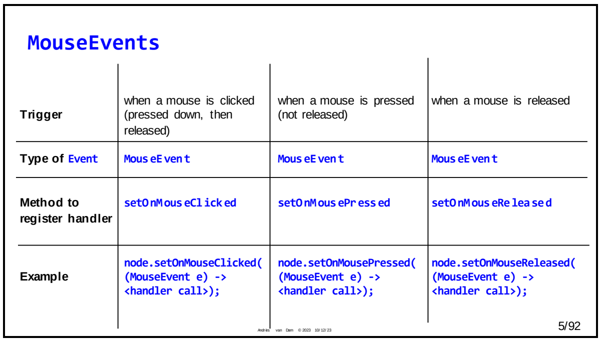
handler call



Mouse Event Handling Example

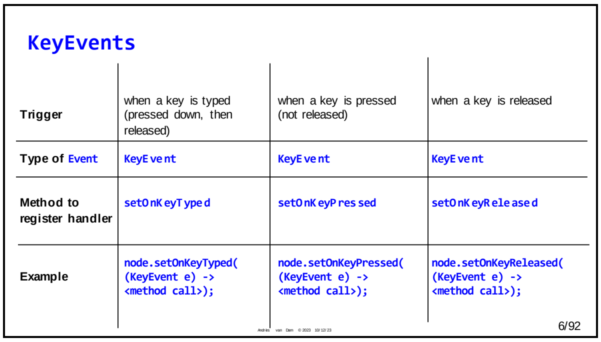
- Let’s say we want our program to respond when you click a circle by printing to the terminal the X and Y locations of the mouse click
- To register a mouse click, we use **setOnMouseClicked**, which requires an **event handler** specialized to a **<MouseEvent>**, written as the type of the first parameter in a lambda expression
- When the mouse is clicked, JavaFX will generate a **MouseEvent**, a bundle of data about that click, and provides get’ers to access it
    - that bundle of data includes the (X, Y) location of the click, which we can retrieve using the **getX** and **getY** method

```
myCircle.setOnMouseClicked((MouseEvent e) ->
                 System.out.println(e.getX() + “, ” + e.getY());
```



MouseEvents

Trigger	when a mouse is clicked (pressed down, then released)	when a mouse is pressed (not released)	when a mouse is released
Type of Event	MouseEvent	MouseEvent	MouseEvent
Method to register handler	setOnMouseClicked	setOnMousePressed	setOnMouseReleased
Example	node.setOnMouseClicked((MouseEvent e) -> <handler call>);	node.setOnMousePressed((MouseEvent e) -> <handler call>);	node.setOnMouseReleased((MouseEvent e) -> <handler call>);



KeyEvents

Trigger	when a key is typed (pressed down, then released)	when a key is pressed (not released)	when a key is released
Type of Event	KeyEvent	KeyEvent	KeyEvent
Method to register handler	setOnKeyTyped	setOnKeyPressed	setOnKeyReleased
Example	node.setOnKeyTyped((KeyEvent e) -> <method call>);	node.setOnKeyPressed((KeyEvent e) -> <method call>);	node.setOnKeyReleased((KeyEvent e) -> <method call>);

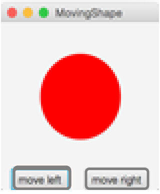
Outline

- Example: MovingShape
- BorderPane
- Constants
- Composite Shapes
  - example: MovingAlien
- Cartoon

Example: MovingShapeApp

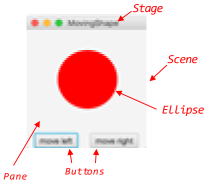
- Program Specification: App that displays a shape and buttons that shift position of the shape left and right by a fixed increment

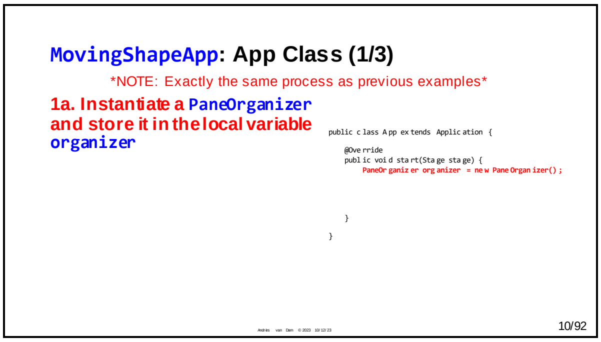
- Purpose: Practice working with absolute positioning of Panes, various Shapes, and more event handling!



Process: MovingShapeApp

1. **Write an 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; ShapeMover will add buttons
3. Write a ShapeMover class which will be responsible for shape movement and other logic. It is instantiated in the PaneOrganizer's constructor
4. Write setupShape() and setupButtons() helper methods to be called within ShapeMover's constructor. These will factor out code for modifying our sub-Panes
5. Register Buttons with **event handlers** that handle Buttons' ActionEvents (clicks) by moving Shape correspondingly, within the ShapeMover class





MovingShapeApp: App Class (1/3)

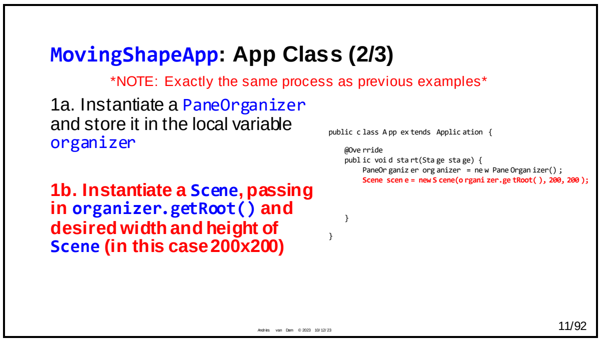
NOTE: Exactly the same process as previous examples

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

```
public class App extends Application {

    @Override
    public void start(Stage stage) {
        PaneOrganizer organizer = new PaneOrganizer();


    }
}
```

Andries van Dam © 2023 10/12/23

10/92



MovingShapeApp: App Class (2/3)

NOTE: Exactly the same process as previous examples

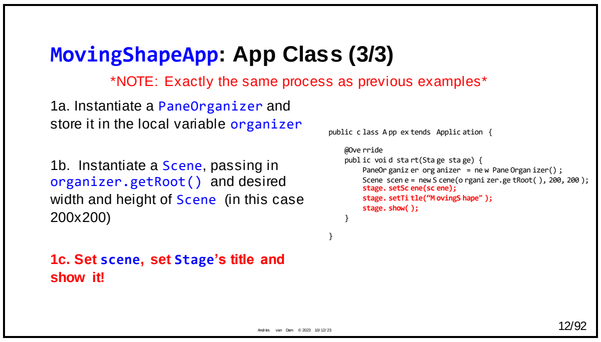
1a. Instantiate a PaneOrganizer and store it in the local variable organizer

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

```
public class App extends Application {

    @Override
    public void start(Stage stage) {
        PaneOrganizer organizer = new PaneOrganizer();
        Scene scene = new Scene(organizer.getRoot(), 200, 200);

    }
}
```

Andries van Dam © 2023 10/12/23

11/92



MovingShapeApp: App Class (3/3)

NOTE: Exactly the same process as previous examples

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

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

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

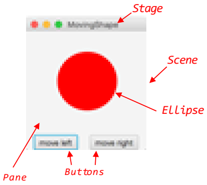
```
public class App extends Application {

    @Override
    public void start(Stage stage) {
        PaneOrganizer organizer = new PaneOrganizer();
        Scene scene = new Scene(organizer.getRoot(), 200, 200);
        stage.setScene(scene);
        stage.setTitle(“MovingShape");
        stage.show();
    }

}
```

Andries van Dam © 2023 10/12/23

12/92

Process: MovingShapeApp

1. Write an `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 all necessary `Pane`s and initialize the `ShapeMover` class**
3. Write a `ShapeMover` class which will be responsible for shapes creation, movement, and other logic. It is instantiated in the `PaneOrganizer`'s constructor
4. Write `setupShape()` and `setupButtons()` helper methods to be called within `ShapeMover`'s constructor. These will factor out code for modifying our sub-`Pane`s
5. Register `Button`s with **`event handlers`** that handle `Button`s' `ActionEvent`s (clicks) by moving `Shape` correspondingly, within the `ShapeMover` class



MovingShapeApp: PaneOrganizer Class (1/3)

2a. Instantiate the root `Pane` and store it in the instance variable `root`

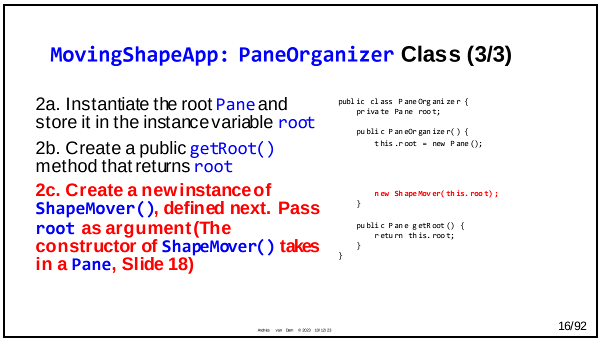
```
public class PaneOrganizer {
    private Pane root;

    public PaneOrganizer() {
        this.root = new Pane();



    }


}
```

MovingShapeApp: PaneOrganizer Class (2/3)

2a. Instantiate the root `Pane` and store it in the instance variable `root`

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

```
public class PaneOrganizer {
    private Pane root;

    public PaneOrganizer() {
        this.root = new Pane();



    }

    public Pane getRoot() {
        return this.root;
    }
}
```



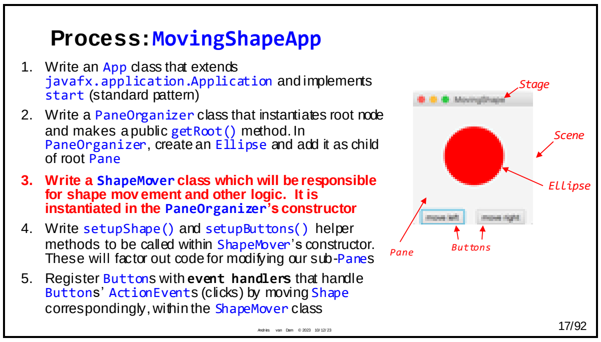
MovingShapeApp: PaneOrganizer Class (3/3)

2a. Instantiate the root Pane and store it in the instance variable root

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

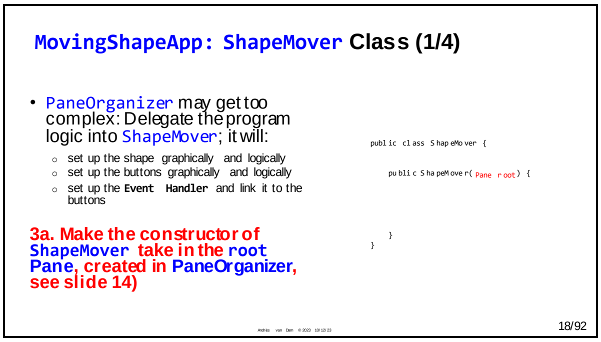
2c. Create a new instance of ShapeMover(), defined next. Pass root as argument (The constructor of ShapeMover() takes in a Pane, Slide 18)

```
public class PaneOrganizer {
    private Pane root;

    public PaneOrganizer() {
        this.root = new Pane();



        new ShapeMover(this.root);
    }

    public Pane getRoot() {
        return this.root;
    }
}
```



Process: MovingShapeApp

1. Write an 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 a ShapeMover class which will be responsible for shape movement and other logic. It is instantiated in the PaneOrganizer's constructor**
4. Write setupShape() and setupButtons() helper methods to be called within ShapeMover's constructor. These will factor out code for modifying our sub-Panes
5. Register Buttons with **event handlers** that handle Buttons' ActionEvents (clicks) by moving Shape correspondingly, within the ShapeMover class

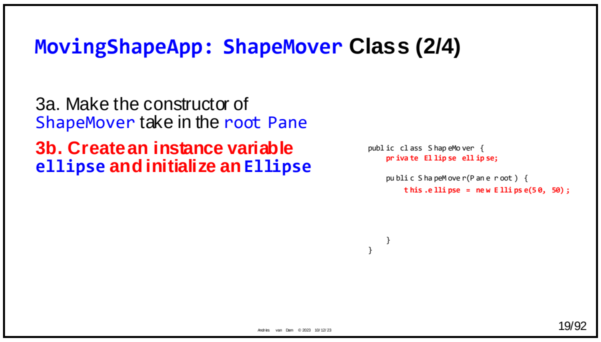


MovingShapeApp: ShapeMover Class (1/4)

- PaneOrganizer may get too complex: Delegate the program logic into ShapeMover; it will:
  - set up the shape graphically and logically
  - set up the buttons graphically and logically
  - set up the **Event Handler** and link it to the buttons

3a. Make the constructor of ShapeMover take in the root Pane, created in PaneOrganizer, see slide 14)

```
public class ShapeMover {

    public ShapeMover(Pane root) {



    }
}
```

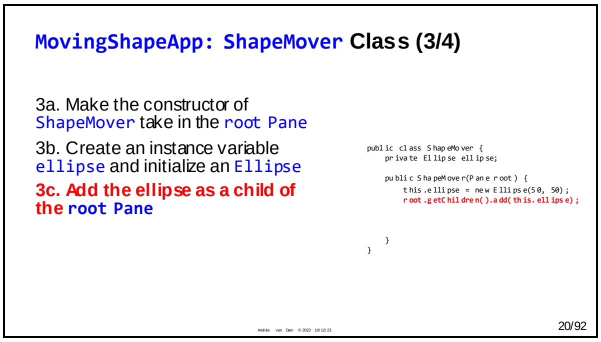


MovingShapeApp: ShapeMover Class (2/4)

3a. Make the constructor of `ShapeMover` take in the `root Pane`

3b. Create an instance variable `ellipse` and initialize an `Ellipse`

```
public class ShapeMover {
    private Ellipse ellipse;

    public ShapeMover(Pane root) {
        this.ellipse = new Ellipse(50, 50);



    }
}
```



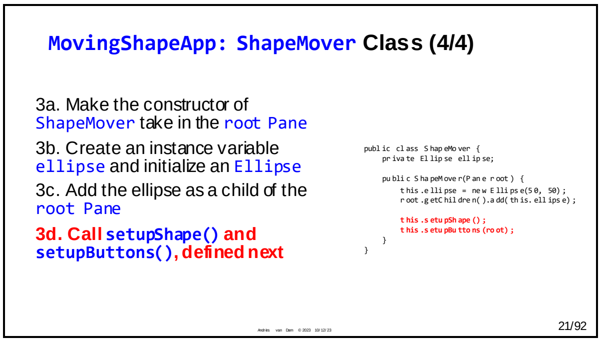
MovingShapeApp: ShapeMover Class (3/4)

3a. Make the constructor of `ShapeMover` take in the `root Pane`

3b. Create an instance variable `ellipse` and initialize an `Ellipse`

3c. Add the ellipse as a child of the `root Pane`

```
public class ShapeMover {
    private Ellipse ellipse;

    public ShapeMover(Pane root) {
        this.ellipse = new Ellipse(50, 50);
        root.getChildren().add(this.ellipse);


    }
}
```



MovingShapeApp: ShapeMover Class (4/4)

3a. Make the constructor of `ShapeMover` take in the `root Pane`

3b. Create an instance variable `ellipse` and initialize an `Ellipse`

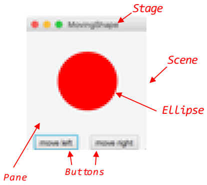
3c. Add the ellipse as a child of the `root Pane`

3d. Call `setupShape()` and `setupButtons()`, defined next

```
public class ShapeMover {
    private Ellipse ellipse;

    public ShapeMover(Pane root) {
        this.ellipse = new Ellipse(50, 50);
        root.getChildren().add(this.ellipse);

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

Process: MovingShapeApp

1. Write an 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 first child of root Pane; ShapeMover will add buttons
3. Write a ShapeMover class which will be responsible for shape movement and other logic. It is instantiated in the PaneOrganizer's constructor
4. **Write setupShape() and setupButtons() helper methods to be called within ShapeMover's constructor. These will factor out code for modifying our sub-Panes**
5. Register Buttons with **event handlers** that handle Buttons' ActionEvents (clicks) by moving Shape correspondingly, within the ShapeMover class



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 ShapeMover constructor—it is good practice to **factor** out code into **helper methods** that are called within the constructor—another use of the **delegation pattern** (which we first used to factor ShapeMover out of PaneOrganizer)
  - setupShape() fills and positions Ellipse
  - setupButtons() adds and positions Buttons, and registers them with their appropriate **event handlers**
- 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 ShapeMover class should be allowed to initialize the color and location of the Ellipse
  - private methods, like private instance variables, are only pseudo-inherited and are therefore not accessible to any external classes or even subclasses—though inherited superclass methods may make use of them w/o the subclasses knowing about them!

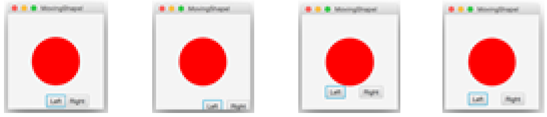
```
public class ShapeMover {
    private Ellipse ellipse;

    public ShapeMover(Pane root) {
        this.ellipse = new Ellipse(50, 50);
        root.getChildren().add(this.ellipse);

        this.setupShape();
        this.setupButtons(root);
    }
    private void setupShape() {
        this.ellipse.setFill(Color.RED);
        this.ellipse.setCenterX(50);
        this.ellipse.setCenterY(50);
    }
}
```

Outline

- Example: MovingShape
- BorderPane
- Constants
- Composite Shapes
  - example: MovingAlien
- Cartoon

Aside: BorderPane 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!



```
left.relocate(50,165);
right.relocate(120,165);
```

```
left.relocate(100,180);
right.relocate(150,180);
```

```
left.relocate(50,150);
right.relocate(120,150);
```

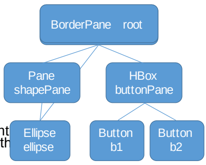
```
left.relocate(50,165);
right.relocate(120,165);
```

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

Aside: BorderPane 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!

Scene graph hierarchy



Aside: BorderPane Class (3/3)

- This makes use of the built-in layout capabilities available to us in JavaFX!
- BorderPane 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 (1/2)

4a. Change root to a BorderPane

```
public class PaneOrganizer {
    private BorderPane root;

    public PaneOrganizer() {
        this.root = new BorderPane();




        new ShapeMover(this.root);
    }

    public Pane getRoot() {
        return this.root;
    }
}
```

MovingShapeApp: update to BorderPane (2/2)

4a. Change root to a BorderPane

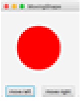
4b. Create a Pane to contain Ellipse. Add shapePane to center of BorderPane by calling setCenter(shapePane) on root

```
public class PaneOrganizer {
    private BorderPane root;

    public PaneOrganizer() {
        this.root = new BorderPane();

        // setup shape pane
        Pane shapePane = new Pane();
        this.root.setCenter(shapePane);



        new ShapeMover(this.root);
    }

    public Pane getRoot() {
        return this.root;
    }
}
```

MovingShapeApp: creation of ButtonPane (1/2)

4c. Instantiate a new HBox, then add it as child of BorderPane, in bottom position

```
public class PaneOrganizer {
    private BorderPane root;

    public PaneOrganizer() {
        this.root = new BorderPane();

        // setup shape pane
        Pane shapePane = new Pane();
        this.root.setCenter(shapePane);

        HBox buttonPane = new HBox();
        this.root.setBottom(buttonPane);

        new ShapeMover(this.root);
    }

    public Pane getRoot() {
        return this.root;
    }
}
```

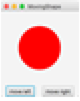


MovingShapeApp: creation of ButtonPane (2/2)

4c. Instantiate a new HBox, then add it as child of BorderPane, in bottom position

4d. Modify the argument of ShapeMover to take in the shapePane and the buttonPane instead of the root Pane

```
public class PaneOrganizer {
    private BorderPane root;

    public PaneOrganizer() {
        this.root = new BorderPane();

        // setup shape pane
        Pane shapePane = new Pane();
        this.root.setCenter(shapePane);

        HBox buttonPane = new HBox();
        this.root.setBottom(buttonPane);

        new ShapeMover( shapePane, buttonPane);
    }

    public Pane getRoot() {
        return this.root;
    }
}
```

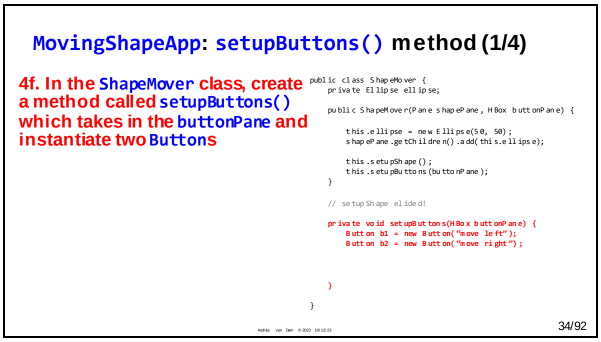


MovingShapeApp: Ellipse in the shapePane

4e. In the ShapeMover class, add the ellipse as a child of the shapePane instead of root

- note: none of the code in our setupShape() 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)
- ShapeMover 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

```
public class ShapeMover {
    private Ellipse ellipse;

    public ShapeMover(Pane shapePane, HBox buttonPane) {

        this.ellipse = new Ellipse(50, 50);
        shapePane.getChildren().add(this.ellipse);

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

    /* setupShape elided! This method sets the color and
     * initial position of the ellipse
     */
}
```



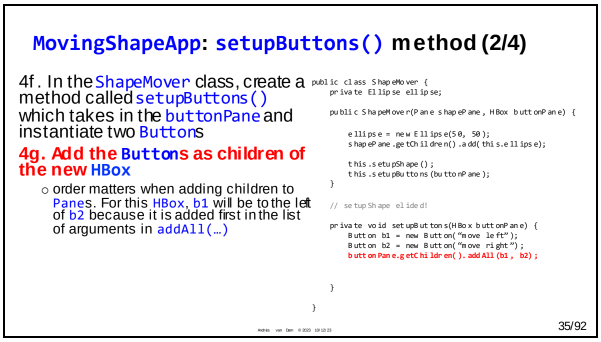
MovingShapeApp: setupButtons() method (1/4)

4f. In the ShapeMover class, create a method called setupButtons() which takes in the buttonPane and instantiate two Buttons

```
public class ShapeMover {
    private Ellipse ellipse;

    public ShapeMover(Pane shapePane, HBox buttonPane) {

        this.ellipse = new Ellipse(50, 50);
        shapePane.getChildren().add(this.ellipse);

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

    // setupShape elided!

    private void setupButtons(HBox buttonPane) {
        Button b1 = new Button(“move left”);
        Button b2 = new Button(“move right”);



    }

}
```

Andries van Dam © 2023 10/12/23

34/92



MovingShapeApp: setupButtons() method (2/4)

4f. In the ShapeMover class, create a method called setupButtons() which takes in the buttonPane and instantiate two Buttons

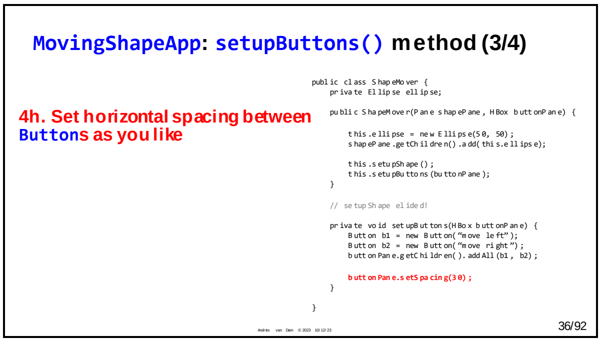
4g. Add the Buttons as children of the new HBox

- ○ order matters when adding children to Panes. For this HBox, b1 will be to the left of b2 because it is added first in the list of arguments in addAll(…)

```
public class ShapeMover {
    private Ellipse ellipse;

    public ShapeMover(Pane shapePane, HBox buttonPane) {

        ellipse = new Ellipse(50, 50);
        shapePane.getChildren().add(this.ellipse);

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

    // setupShape elided!

    private void setupButtons(HBox buttonPane) {
        Button b1 = new Button(“move left”);
        Button b2 = new Button(“move right”);
        buttonPane.getChildren().addAll(b1, b2);


    }

}
```

Andries van Dam © 2023 10/12/23

35/92



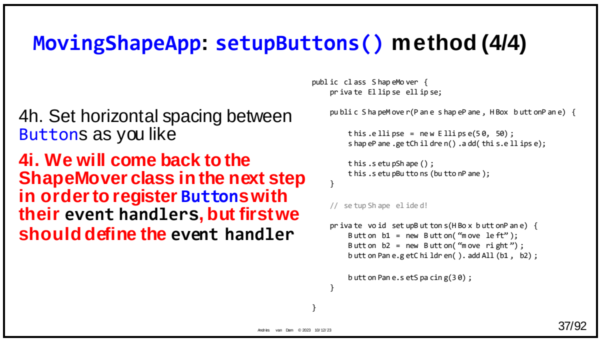
MovingShapeApp: setupButtons() method (3/4)

4h. Set horizontal spacing between Buttons as you like

```
public class ShapeMover {
    private Ellipse ellipse;

    public ShapeMover(Pane shapePane, HBox buttonPane) {

        this.ellipse = new Ellipse(50, 50);
        shapePane.getChildren().add(this.ellipse);

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

    // setupShape elided!

    private void setupButtons(HBox buttonPane) {
        Button b1 = new Button(“move left”);
        Button b2 = new Button(“move right”);
        buttonPane.getChildren().addAll(b1, b2);

        buttonPane.setSpacing(30);
    }

}
```

Andries van Dam © 2023 10/12/23

36/92

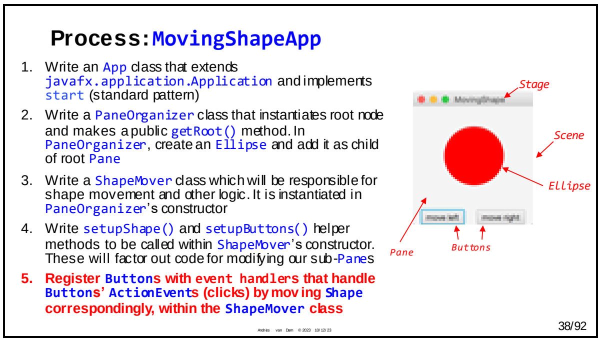


MovingShapeApp: setupButtons() method (4/4)

4h. Set horizontal spacing between Buttons as you like

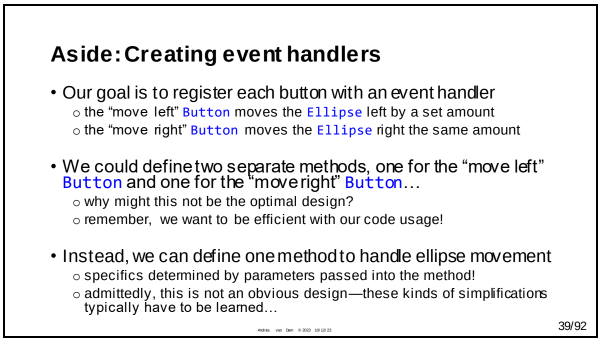
4i. We will come back to the ShapeMover class in the next step in order to register Buttons with their event handlers, but first we should define the event handler

```
public class ShapeMover {
    private Ellipse ellipse;

    public ShapeMover(Pane shapePane, HBox buttonPane) {

        this.ellipse = new Ellipse(50, 50);
        shapePane.getChildren().add(this.ellipse);

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

    // setupShape elided!

    private void setupButtons(HBox buttonPane) {
        Button b1 = new Button("move left");
        Button b2 = new Button("move right");
        buttonPane.getChildren().addAll(b1, b2);

        buttonPane.setSpacing(30);
    }

}
```



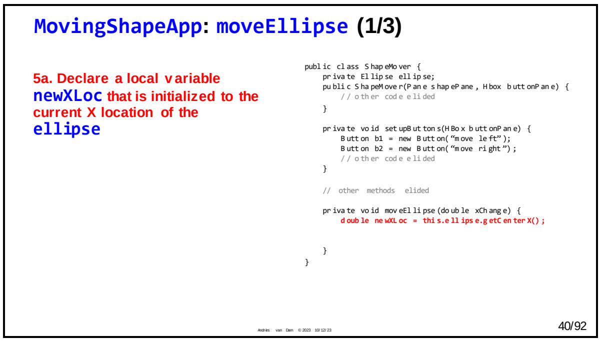
Process: MovingShapeApp

1. Write an 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 a ShapeMover class which will be responsible for shape movement and other logic. It is instantiated in PaneOrganizer's constructor
4. Write setupShape() and setupButtons() helper methods to be called within ShapeMover's constructor. These will factor out code for modifying our sub-Panes
5. **Register Buttons with event handlers that handle Buttons' ActionEvents (clicks) by moving Shape correspondingly, within the ShapeMover class**



Aside: Creating event handlers

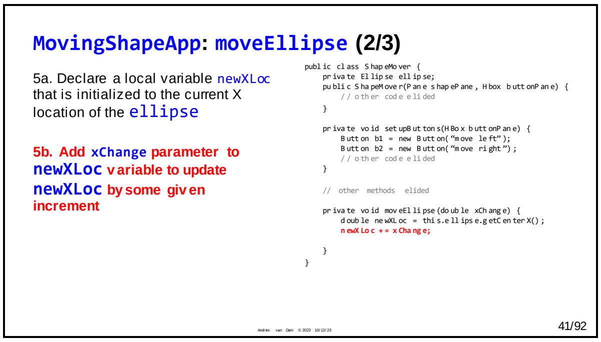
- Our goal is to register each button with an event handler
  - the "move left" Button moves the Ellipse left by a set amount
  - the "move right" Button moves the Ellipse right the same amount
- We could define two separate methods, one for the "move left" Button and one for the "move right" Button…
  - why might this not be the optimal design?
  - remember, we want to be efficient with our code usage!
- Instead, we can define one method to handle ellipse movement
  - specifics determined by parameters passed into the method!
  - admittedly, this is not an obvious design—these kinds of simplifications typically have to be learned…



MovingShapeApp: moveEllipse (1/3)

5a. Declare a local variable newXLoc that is initialized to the current X location of the ellipse

```
public class ShapeMover {
    private Ellipse ellipse;
    public ShapeMover(Pane shapePane, Hbox buttonPane) {
        // other code elided
    }

    private void setupButtons(HBox buttonPane) {
        Button b1 = new Button(“move left”);
        Button b2 = new Button(“move right”);
        // other code elided
    }

    // other methods elided

    private void moveEllipse(double xChange) {
        double newXLoc = this.ellipse.getCenterX();


    }
}
```

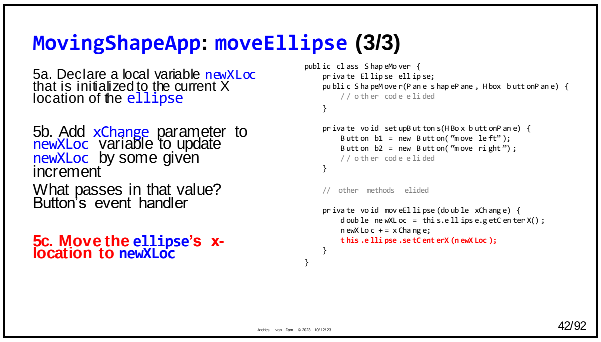


MovingShapeApp: moveEllipse (2/3)

5a. Declare a local variable newXLoc that is initialized to the current X location of the ellipse

5b. Add xChange parameter to newXLoc variable to update newXLoc by some given increment

```
public class ShapeMover {
    private Ellipse ellipse;
    public ShapeMover(Pane shapePane, Hbox buttonPane) {
        // other code elided
    }

    private void setupButtons(HBox buttonPane) {
        Button b1 = new Button(“move left”);
        Button b2 = new Button(“move right”);
        // other code elided
    }

    // other methods elided

    private void moveEllipse(double xChange) {
        double newXLoc = this.ellipse.getCenterX();
        newXLoc += xChange;

    }
}
```



MovingShapeApp: moveEllipse (3/3)

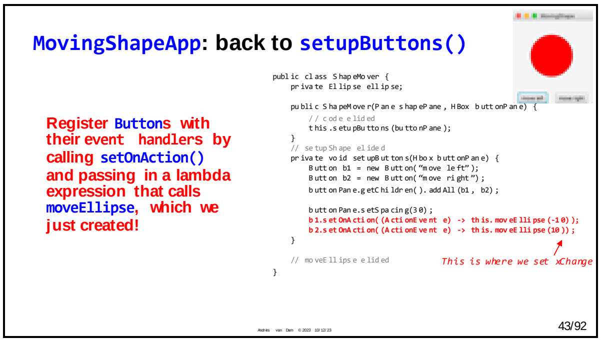
5a. Declare a local variable newXLoc that is initialized to the current X location of the ellipse

5b. Add xChange parameter to newXLoc variable to update newXLoc by some given increment

What passes in that value? Button's event handler

5c. Move the ellipse's x-location to newXLoc

```
public class ShapeMover {
    private Ellipse ellipse;
    public ShapeMover(Pane shapePane, Hbox buttonPane) {
        // other code elided
    }

    private void setupButtons(HBox buttonPane) {
        Button b1 = new Button(“move left”);
        Button b2 = new Button(“move right”);
        // other code elided
    }

    // other methods elided

    private void moveEllipse(double xChange) {
        double newXLoc = this.ellipse.getCenterX();
        newXLoc += xChange;
        this.ellipse.setCenterX(newXLoc);
    }
}
```



MovingShapeApp: back to setupButtons()

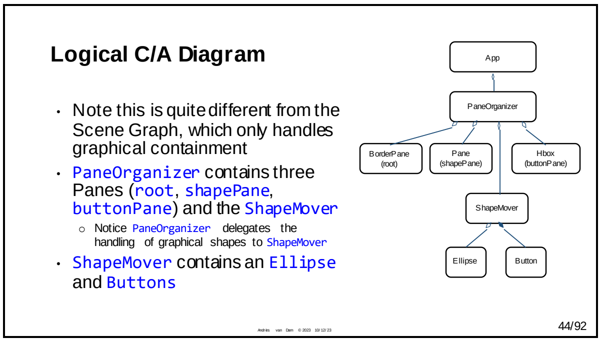
Register Buttons with their event handlers by calling setOnAction() and passing in a lambda expression that calls moveEllipse, which we just created!

```
public class ShapeMover {
    private Ellipse ellipse;

    public ShapeMover(Pane shapePane, HBox buttonPane) {
        // code elided
        this.setupButtons(buttonPane);
    }
    // setupShape elided
    private void setupButtons(Hbox buttonPane) {
        Button b1 = new Button(“move left”);
        Button b2 = new Button(“move right”);
        buttonPane.getChildren().addAll(b1, b2);

        buttonPane.setSpacing(30);
        b1.setOnAction((ActionEvent e) -> this.moveEllipse(-10));
        b2.setOnAction((ActionEvent e) -> this.moveEllipse(10));
    }

    // moveEllipse elided
}
```

This is where we set xChange

Andries van Dam © 2023 10/12/23

43/92



Logical C/A Diagram

- Note this is quite different from the Scene Graph, which only handles graphical containment
- PaneOrganizer contains three Panes (root, shapePane, buttonPane) and the ShapeMover
  - Notice PaneOrganizer delegates the handling of graphical shapes to ShapeMover
- ShapeMover contains an Ellipse and Buttons

App

PaneOrganizer

BorderPane (root)

Pane (shapePane)

Hbox (buttonPane)

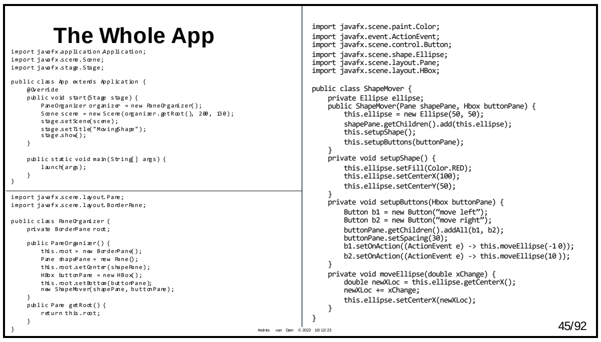
ShapeMover

Ellipse

Button

Andries van Dam © 2023 10/12/23

44/92



The Whole App

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;

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);
    }
}
```

```
import javafx.scene.layout.Pane;
import javafx.scene.layout.BorderPane;

public class PaneOrganizer {
    private BorderPane root;

    public PaneOrganizer() {
        this.root = new BorderPane();
        Pane shapePane = new Pane();
        this.root.setCenter(shapePane);
        HBox buttonPane = new HBox();
        this.root.setBottom(buttonPane);
        new ShapeMover(shapePane, buttonPane);
    }
    public Pane getRoot() {
        return this.root;
    }
}
```

```
import javafx.scene.paint.Color;
import javafx.event.ActionEvent;
import javafx.scene.control.Button;
import javafx.scene.shape.Ellipse;
import javafx.scene.layout.Pane;
import javafx.scene.layout.HBox;

public class ShapeMover {
    private Ellipse ellipse;
    public ShapeMover(Pane shapePane, Hbox buttonPane) {
        this.ellipse = new Ellipse(50, 50);
        shapePane.getChildren().add(this.ellipse);
        this.setupShape();
        this.setupButtons(buttonPane);
    }
    private void setupShape() {
        this.ellipse.setFill(Color.RED);
        this.ellipse.setCenterX(100);
        this.ellipse.setCenterY(50);
    }
    private void setupButtons(Hbox buttonPane) {
        Button b1 = new Button(“move left”);
        Button b2 = new Button(“move right”);
        buttonPane.getChildren().addAll(b1, b2);
        buttonPane.setSpacing(30);
        b1.setOnAction((ActionEvent e) -> this.moveEllipse(-10));
        b2.setOnAction((ActionEvent e) -> this.moveEllipse(10));
    }
    private void moveEllipse(double xChange) {
        double newXLoc = this.ellipse.getCenterX();
        newXLoc += xChange;
        this.ellipse.setCenterX(newXLoc);
    }
}
```

Andries van Dam © 2023 10/12/23

45/92

Outline

- Example: MovingShape
- BorderPane
- Constants
- Composite Shapes
  - example: MovingAlien
- Cartoon

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
- For our MovingShapeApp, make constants for width and height of the Ellipse and of the Pane it sits in, as well as the start location and distance moved

```
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;
}
```

TopHat Question

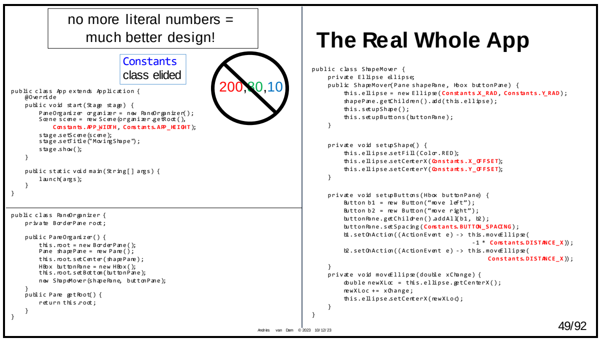
When should you define a value in a Constants class?

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.



no more literal numbers = much better design!

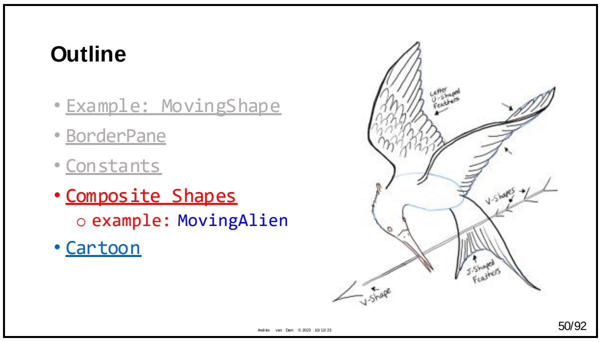
Constants class elided

```
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);
    }
}
```

```
public class PaneOrganizer {
    private BorderPane root;

    public PaneOrganizer() {
        this.root = new BorderPane();
        Pane shapePane = new Pane();
        this.root.setCenter(shapePane);
        HBox buttonPane = new HBox();
        this.root.setBottom(buttonPane);
        new ShapeMover(shapePane, buttonPane);
    }
    public Pane getRoot() {
        return this.root;
    }
}
```

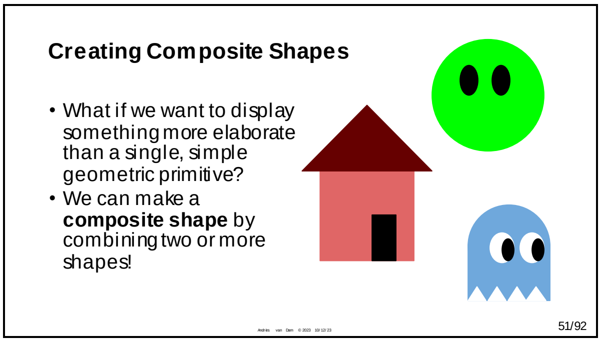
The Real Whole App

```
public class ShapeMover {
    private Ellipse ellipse;
    public ShapeMover(Pane shapePane, Hbox buttonPane) {
        this.ellipse = new Ellipse(Constants.X_RAD, Constants.Y_RAD);
        shapePane.getChildren().add(this.ellipse);
        this.setupShape();
        this.setupButtons(buttonPane);
    }


    private void setupShape() {
        this.ellipse.setFill(Color.RED);
        this.ellipse.setCenterX(Constants.X_OFFSET);
        this.ellipse.setCenterY(Constants.Y_OFFSET);
    }

    private void setupButtons(Hbox buttonPane) {
        Button b1 = new Button(“move left”);
        Button b2 = new Button(“move right”);
        buttonPane.getChildren().addAll(b1, b2);
        buttonPane.setSpacing(Constants.BUTTON_SPACING);
        b1.setOnAction((ActionEvent e) -> this.moveEllipse(
                                    -1 * Constants.DISTANCE_X));
        b2.setOnAction((ActionEvent e) -> this.moveEllipse(
                                       Constants.DISTANCE_X));
    }
    private void moveEllipse(double xChange) {
        double newXLoc = this.ellipse.getCenterX();
        newXLoc += xChange;
        this.ellipse.setCenterX(newXLoc);
    }
}
```



Outline

- Example: MovingShape
- BorderPane
- Constants
- Composite Shapes
  - example: MovingAlien
- Cartoon

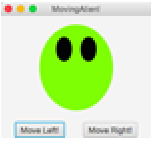


Creating Composite Shapes

- 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!

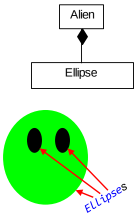
Specifications: MovingAlien

- 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 AlienMover (which used to be ShapeMover)!
3. Modify AlienMover to contain an Alien instead of an Ellipse

Alien Class

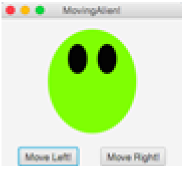
- The Alien class is our composite shape
- It contains three Ellipses—one for the face and one for each eye
- Constructor instantiates these Ellipses, sets their initial sizes/colors, and adds them as children of the alienPane—which was passed in as a parameter
- Although Alien class deals with each 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 Alien class to define methods for manipulating composite shape(s) in Pane

```
public class Alien {
    private Ellipse face;
    private Ellipse leftEye;
    private Ellipse rightEye;

    public Alien( Pane alienPane) {//Alien lives in passed Pane
        this.face = new Ellipse(Constants.X_RAD, Constants.Y_RAD);
        this.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
        in the program.*/

        this.leftEye = new Ellipse(Constants.EYE_X, Constants.EYE_Y);
        this.leftEye.setFill(Color.BLACK);
        this.rightEye = new Ellipse(Constants.EYE_X, Constants.EYE_Y);
        this.rightEye.setFill(Color.BLACK);

        alienPane.getChildren().addAll(this.face, this.leftEye,
                                       this.rightEye);

    }
}
```

Note: Order matters when you add children to a Pane! The arguments are added in that order graphically and if there is overlap, the shape later in the parameter list will lie wholly or partially on top of the earlier one. For this example, face is added first, then leftEye and rightEye on top. The inverse order would be wrong!

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 AlienMover (which used to be ShapeMover)!**
3. Modify AlienMover to contain an Alien instead of an Ellipse

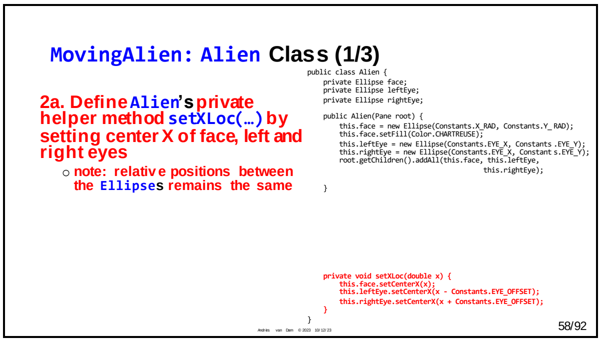


Alien Class

- In MovingShapeApp, the following call is made from within our moveEllipse method:

```
this.ellipse.setCenterX(newXLoc);
```

- Because we called JavaFX's getCenterX() and setCenterX(…) on our shape from within the ShapeMover class, we must now define our own methods to set the Alien's location in the Alien class!
- Keep it simple: what are the **capabilities** (methods) we want the Alien to have?
  - move left
  - move right
- As earlier, moveLeft and moveRight will share some code, so we can use a private helper method

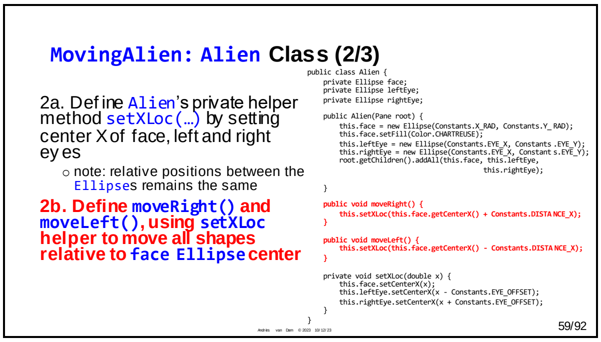


MovingAlien: Alien Class (1/3)

2a. Define Alien’s private helper method setXLoc(…) by setting center X of face, left and right eyes

- **note: relative positions between the Ellipses remains the same**

```
public class Alien {
    private Ellipse face;
    private Ellipse leftEye;
    private Ellipse rightEye;

    public Alien(Pane root) {
        this.face = new Ellipse(Constants.X_RAD, Constants.Y_RAD);
        this.face.setFill(Color.CHARTREUSE);
        this.leftEye = new Ellipse(Constants.EYE_X, Constants.EYE_Y);
        this.rightEye = new Ellipse(Constants.EYE_X, Constants.EYE_Y);
        root.getChildren().addAll(this.face, this.leftEye,
                                    this.rightEye);

    }

    private void setXLoc(double x) {
        this.face.setCenterX(x);
        this.leftEye.setCenterX(x - Constants.EYE_OFFSET);
        this.rightEye.setCenterX(x + Constants.EYE_OFFSET);
    }
}
```



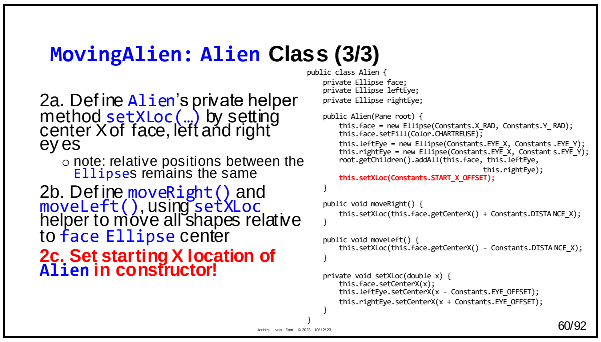
MovingAlien: Alien Class (2/3)

2a. Define Alien’s private helper method setXLoc(…) by setting center X of face, left and right eyes

- note: relative positions between the Ellipses remains the same

2b. Define moveRight() and moveLeft(), using setXLoc helper to move all shapes relative to face Ellipse center

```
public class Alien {
    private Ellipse face;
    private Ellipse leftEye;
    private Ellipse rightEye;

    public Alien(Pane root) {
        this.face = new Ellipse(Constants.X_RAD, Constants.Y_RAD);
        this.face.setFill(Color.CHARTREUSE);
        this.leftEye = new Ellipse(Constants.EYE_X, Constants.EYE_Y);
        this.rightEye = new Ellipse(Constants.EYE_X, Constants.EYE_Y);
        root.getChildren().addAll(this.face, this.leftEye,
                                    this.rightEye);

    }

    public void moveRight() {
        this.setXLoc(this.face.getCenterX() + Constants.DISTANCE_X);
    }

    public void moveLeft() {
        this.setXLoc(this.face.getCenterX() - Constants.DISTANCE_X);
    }

    private void setXLoc(double x) {
        this.face.setCenterX(x);
        this.leftEye.setCenterX(x - Constants.EYE_OFFSET);
        this.rightEye.setCenterX(x + Constants.EYE_OFFSET);
    }
}
```



MovingAlien: Alien Class (3/3)

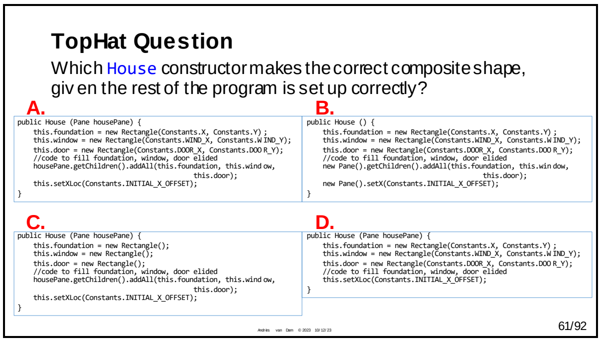
2a. Define Alien’s private helper method setXLoc(…) by setting center X of face, left and right eyes

- note: relative positions between the Ellipses remains the same

2b. Define moveRight() and moveLeft(), using setXLoc helper to move all shapes relative to face Ellipse center

2c. Set starting X location of Alien in constructor!

```
public class Alien {
    private Ellipse face;
    private Ellipse leftEye;
    private Ellipse rightEye;

    public Alien(Pane root) {
        this.face = new Ellipse(Constants.X_RAD, Constants.Y_RAD);
        this.face.setFill(Color.CHARTREUSE);
        this.leftEye = new Ellipse(Constants.EYE_X, Constants.EYE_Y);
        this.rightEye = new Ellipse(Constants.EYE_X, Constants.EYE_Y);
        root.getChildren().addAll(this.face, this.leftEye,
                                    this.rightEye);
        this.setXLoc(Constants.START_X_OFFSET);
    }

    public void moveRight() {
        this.setXLoc(this.face.getCenterX() + Constants.DISTANCE_X);
    }

    public void moveLeft() {
        this.setXLoc(this.face.getCenterX() - Constants.DISTANCE_X);
    }

    private void setXLoc(double x) {
        this.face.setCenterX(x);
        this.leftEye.setCenterX(x - Constants.EYE_OFFSET);
        this.rightEye.setCenterX(x + Constants.EYE_OFFSET);
    }
}
```



TopHat Question

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

A.

```
public House (Pane housePane) {
    this.foundation = new Rectangle(Constants.X, Constants.Y) ;
    this.window = new Rectangle(Constants.WIND_X, Constants.WIND_Y);
    this.door = new Rectangle(Constants.DOOR_X, Constants.DOOR_Y);
    //code to fill foundation, window, door elided
    housePane.getChildren().addAll(this.foundation, this.window,
                                        this.door);
    this.setXLoc(Constants.INITIAL_X_OFFSET);
}
```

B.

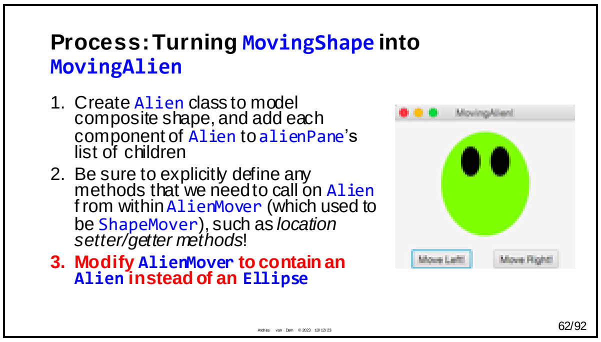
```
public House () {
    this.foundation = new Rectangle(Constants.X, Constants.Y) ;
    this.window = new Rectangle(Constants.WIND_X, Constants.WIND_Y);
    this.door = new Rectangle(Constants.DOOR_X, Constants.DOOR_Y);
    //code to fill foundation, window, door elided
    new Pane().getChildren().addAll(this.foundation, this.window,
                                        this.door);
    new Pane().setX(Constants.INITIAL_X_OFFSET);
}
```

C.

```
public House (Pane housePane) {
    this.foundation = new Rectangle();
    this.window = new Rectangle();
    this.door = new Rectangle();
    //code to fill foundation, window, door elided
    housePane.getChildren().addAll(this.foundation, this.window,
                                        this.door);
    this.setXLoc(Constants.INITIAL_X_OFFSET);
}
```

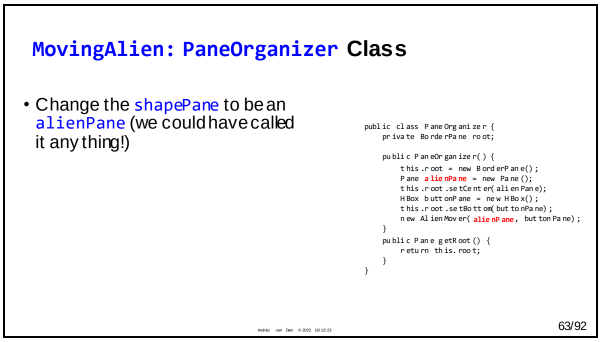
D.

```
public House (Pane housePane) {
    this.foundation = new Rectangle(Constants.X, Constants.Y) ;
    this.window = new Rectangle(Constants.WIND_X, Constants.WIND_Y);
    this.door = new Rectangle(Constants.DOOR_X, Constants.DOOR_Y);
    //code to fill foundation, window, door elided
    this.setXLoc(Constants.INITIAL_X_OFFSET);
}
```



Process: Turning `MovingShape` into `MovingAlien`

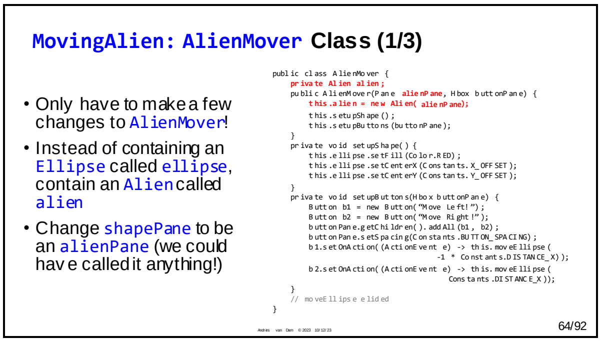
1. Create `Alien` class to model composite shape, and add each component of `Alien` to `alienPane`'s list of children
2. Be sure to explicitly define any methods that we need to call on `Alien` from within `AlienMover` (which used to be `ShapeMover`), such as *location setter/getter methods*!
3. **Modify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



`MovingAlien: PaneOrganizer` Class

- Change the `shapePane` to be an `alienPane` (we could have called it any thing!)

```
public class PaneOrganizer {
    private BorderPane root;

    public PaneOrganizer() {
        this.root = new BorderPane();
        Pane alienPane = new Pane();
        this.root.setCenter(alienPane);
        HBox buttonPane = new HBox();
        this.root.setBottom(buttonPane);
        new AlienMover(alienPane, buttonPane);
    }
    public Pane getRoot() {
        return this.root;
    }
}
```



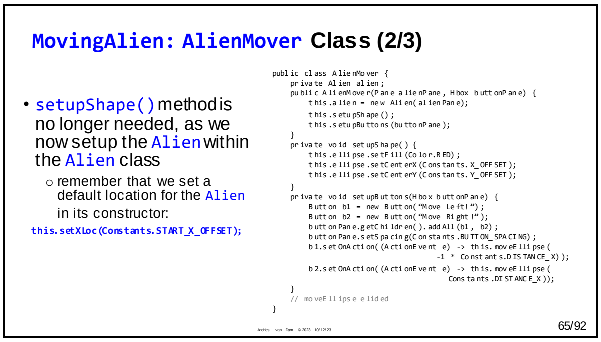
MovingAlien: AlienMover Class (1/3)

- Only have to make a few changes to AlienMover!
- Instead of containing an Ellipse called ellipse, contain an Alien called alien
- Change shapePane to be an alienPane (we could have called it anything!)

```
public class AlienMover {
    private Alien alien;
    public AlienMover(Pane alienPane, Hbox buttonPane) {
        this.alien = new Alien(alienPane);
        this.setupShape();
        this.setupButtons(buttonPane);
    }
    private void setupShape() {
        this.ellipse.setFill(Color.RED);
        this.ellipse.setCenterX(Constants.X_OFFSET);
        this.ellipse.setCenterY(Constants.Y_OFFSET);
    }
    private void setupButtons(Hbox buttonPane) {
        Button b1 = new Button(“Move Left!”);
        Button b2 = new Button(“Move Right!”);
        buttonPane.getChildren().addAll(b1, b2);
        buttonPane.setSpacing(Constants.BUTTON_SPACING);
        b1.setOnAction((ActionEvent e) -> this.moveEllipse(
                                        -1 * Constants.DISTANCE_X));
        b2.setOnAction((ActionEvent e) -> this.moveEllipse(
                                          Constants.DISTANCE_X));
    }
    // moveEllipse elided
}
```

Andries van Dam © 2023 10/12/23

64/92



MovingAlien: AlienMover Class (2/3)

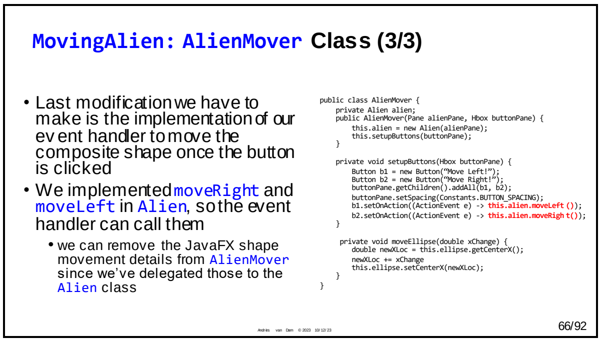
- setupShape() method is no longer needed, as we now setup the Alien within the Alien class
  - remember that we set a default location for the Alien in its constructor:

this.setXLoc(Constants.START_X_OFFSET);

```
public class AlienMover {
    private Alien alien;
    public AlienMover(Pane alienPane, Hbox buttonPane) {
        this.alien = new Alien(alienPane);
        this.setupShape();
        this.setupButtons(buttonPane);
    }
    private void setupShape() {
        this.ellipse.setFill(Color.RED);
        this.ellipse.setCenterX(Constants.X_OFFSET);
        this.ellipse.setCenterY(Constants.Y_OFFSET);
    }
    private void setupButtons(Hbox buttonPane) {
        Button b1 = new Button(“Move Left!”);
        Button b2 = new Button(“Move Right!”);
        buttonPane.getChildren().addAll(b1, b2);
        buttonPane.setSpacing(Constants.BUTTON_SPACING);
        b1.setOnAction((ActionEvent e) -> this.moveEllipse(
                                        -1 * Constants.DISTANCE_X));
        b2.setOnAction((ActionEvent e) -> this.moveEllipse(
                                          Constants.DISTANCE_X));
    }
    // moveEllipse elided
}
```

Andries van Dam © 2023 10/12/23

65/92



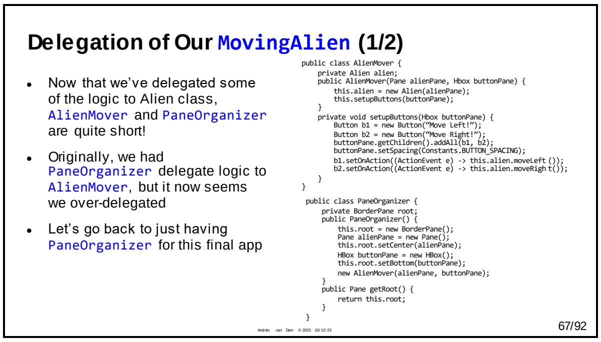
MovingAlien: AlienMover Class (3/3)

- Last modification we have to make is the implementation of our event handler to move the composite shape once the button is clicked
- We implemented moveRight and moveLeft in Alien, so the event handler can call them
  - we can remove the JavaFX shape movement details from AlienMover since we've delegated those to the Alien class

```
public class AlienMover {
    private Alien alien;
    public AlienMover(Pane alienPane, Hbox buttonPane) {
        this.alien = new Alien(alienPane);
        this.setupButtons(buttonPane);
    }

    private void setupButtons(Hbox buttonPane) {
        Button b1 = new Button(“Move Left!”);
        Button b2 = new Button(“Move Right!”);
        buttonPane.getChildren().addAll(b1, b2);
        buttonPane.setSpacing(Constants.BUTTON_SPACING);
        b1.setOnAction((ActionEvent e) -> this.alien.moveLeft());
        b2.setOnAction((ActionEvent e) -> this.alien.moveRight());
    }

     private void moveEllipse(double xChange) {
        double newXLoc = this.ellipse.getCenterX();
        newXLoc += xChange
        this.ellipse.setCenterX(newXLoc);
    }
}
```

Andries van Dam © 2023 10/12/23

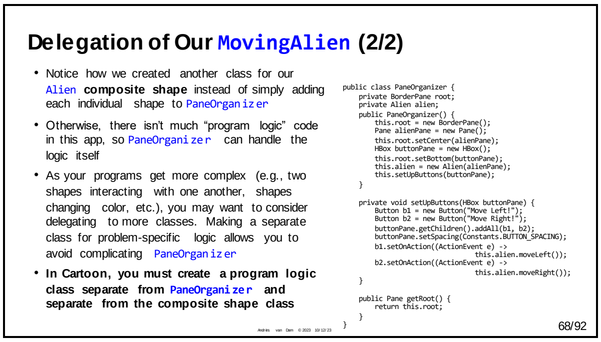
66/92



Delegation of Our `MovingAlien` (1/2)

- Now that we've delegated some of the logic to Alien class, `AlienMover` and `PaneOrganizer` are quite short!
- Originally, we had `PaneOrganizer` delegate logic to `AlienMover`, but it now seems we over-delegated
- Let's go back to just having `PaneOrganizer` for this final app

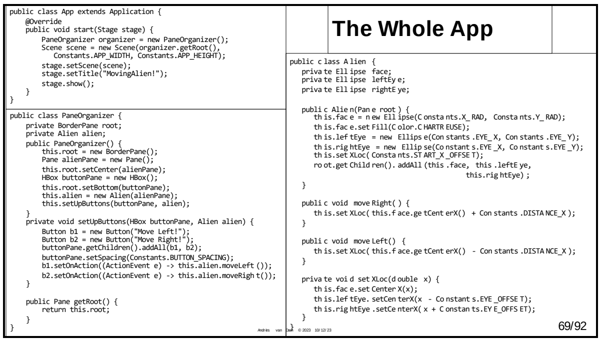
```
public class AlienMover {
    private Alien alien;
    public AlienMover(Pane alienPane, Hbox buttonPane) {
        this.alien = new Alien(alienPane);
        this.setupButtons(buttonPane);
    }
    private void setupButtons(Hbox buttonPane) {
        Button b1 = new Button(“Move Left!”);
        Button b2 = new Button(“Move Right!”);
        buttonPane.getChildren().addAll(b1, b2);
        buttonPane.setSpacing(Constants.BUTTON_SPACING);
        b1.setOnAction((ActionEvent e) -> this.alien.moveLeft ());
        b2.setOnAction((ActionEvent e) -> this.alien.moveRigh t());
    }
}

 public class PaneOrganizer {
     private BorderPane root;
     public PaneOrganizer() {
         this.root = new BorderPane();
         Pane alienPane = new Pane();
         this.root.setCenter(alienPane);
         HBox buttonPane = new HBox();
         this.root.setBottom(buttonPane);
         new AlienMover(alienPane, buttonPane);
     }
     public Pane getRoot() {
         return this.root;
     }
 }
```



Delegation of Our `MovingAlien` (2/2)

- Notice how we created another class for our `Alien` **composite shape** instead of simply adding each individual shape to `PaneOrganizer`
- Otherwise, there isn't much "program logic" code in this app, so `PaneOrganizer` can handle the logic itself
- As your programs get more complex (e.g., two shapes interacting with one another, shapes changing color, etc.), you may want to consider delegating to more classes. Making a separate class for problem-specific logic allows you to avoid complicating `PaneOrganizer`
- **In Cartoon, you must create a program logic class separate from `PaneOrganizer` and separate from the composite shape class**

```
public class PaneOrganizer {
    private BorderPane root;
    private Alien alien;
    public PaneOrganizer() {
        this.root = new BorderPane();
        Pane alienPane = new Pane();
        this.root.setCenter(alienPane);
        HBox buttonPane = new HBox();
        this.root.setBottom(buttonPane);
        this.alien = new Alien(alienPane);
        this.setUpButtons(buttonPane);
    }

    private void setUpButtons(HBox buttonPane) {
        Button b1 = new Button("Move Left!");
        Button b2 = new Button("Move Right!");
        buttonPane.getChildren().addAll(b1, b2);
        buttonPane.setSpacing(Constants.BUTTON_SPACING);
        b1.setOnAction((ActionEvent e) ->
                                     this.alien.moveLeft());
        b2.setOnAction((ActionEvent e) ->
                                     this.alien.moveRight());
    }

    public Pane getRoot() {
        return this.root;
    }
}
```



The Whole App

```
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 class PaneOrganizer {
    private BorderPane root;
    private Alien alien;
    public PaneOrganizer() {
        this.root = new BorderPane();
        Pane alienPane = new Pane();
        this.root.setCenter(alienPane);
        HBox buttonPane = new HBox();
        this.root.setBottom(buttonPane);
        this.alien = new Alien(alienPane);
        this.setUpButtons(buttonPane, alien);
    }
    private void setUpButtons(HBox buttonPane, Alien alien) {
        Button b1 = new Button("Move Left!");
        Button b2 = new Button("Move Right!");
        buttonPane.getChildren().addAll(b1, b2);
        buttonPane.setSpacing(Constants.BUTTON_SPACING);
        b1.setOnAction((ActionEvent e) -> this.alien.moveLeft ());
        b2.setOnAction((ActionEvent e) -> this.alien.moveRigh t());
    }

    public Pane getRoot() {
        return this.root;
    }
}
```

```
public class Alien {
   private Ellipse face;
   private Ellipse leftEye;
   private Ellipse rightEye;

   public Alien(Pane root) {
      this.face = new Ellipse(Constants.X_RAD, Constants.Y_RAD);
      this.face.setFill(Color.CHARTREUSE);
      this.leftEye = new Ellipse(Constants.EYE_X, Constants.EYE_Y);
      this.rightEye = new Ellipse(Constants.EYE_X, Constants.EYE_Y);
      this.setXLoc(Constants.START_X_OFFSET);
      root.getChildren().addAll(this.face, this.leftEye,
                                             this.rightEye);
   }

   public void moveRight() {
      this.setXLoc(this.face.getCenterX() + Constants.DISTANCE_X);
   }

   public void moveLeft() {
      this.setXLoc(this.face.getCenterX() - Constants.DISTANCE_X);
   }

   private void setXLoc(double x) {
      this.face.setCenterX(x);
      this.leftEye.setCenterX(x - Constants.EYE_OFFSET);
      this.rightEye.setCenterX(x + Constants.EYE_OFFSET);
   }
}
```

TopHat 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.

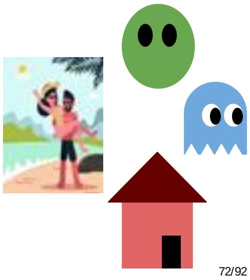
C. Use a top-level class, make classes for more complicated shapes, and store composite shapes, or just generally related objects, within panes.

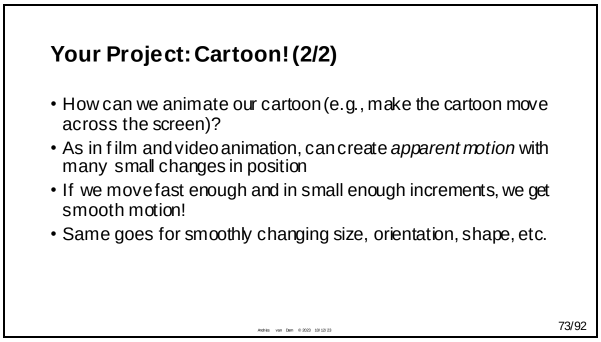
Outline

- Example: MovingShape
- BorderPane
- Constants
- Composite Shapes
  - example: MovingAlien
- Cartoon

Your Project: Cartoon! (1/2)

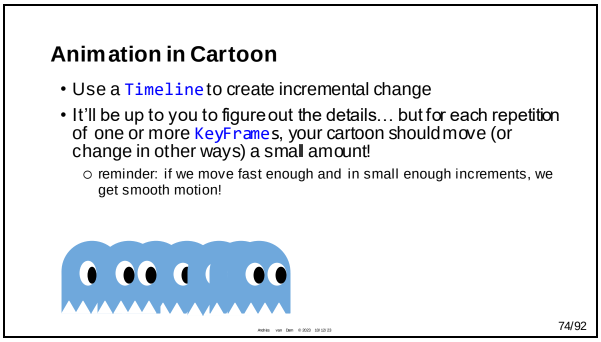
- You'll be building a JavaFX application that displays your own custom "cartoon", much like the examples in this lecture
- But your cartoon will be animated!





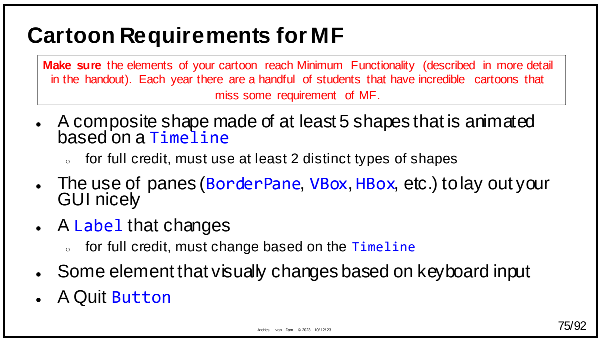
Your Project: Cartoon! (2/2)

- 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 one or more `KeyFrame`s, 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!



Cartoon Requirements for MF

Make sure the elements of your cartoon reach Minimum Functionality (described in more detail in the handout). Each year there are a handful of students that have incredible cartoons that miss some requirement of MF.

- A composite shape made of at least 5 shapes that is animated based on a `Timeline`
  - for full credit, must use at least 2 distinct types of shapes
- The use of panes (`BorderPane`, `VBox`, `HBox`, etc.) to lay out your GUI nicely
- A `Label` that changes
  - for full credit, must change based on the `Timeline`
- Some element that visually changes based on keyboard input
- A Quit `Button`

Cartoon Competition!

- With open-ended project, so much room for “Bells & Whistles” for extra credit!
  - experiment with other fancy JavaFX animation features (fades, path animations, etc.)
  - include other JavaFX elements like `Slider`s, `Spinner`s, and `ColorPicker`s
  - use mouse interaction *and* keyboard interaction
  - add ~ polymorphism ~ (in a meaningful way)
  - anything else you can come up!
- The staff will vote on the top 6 cartoons to enjoy a special lunch with Andy at Kabob & Curry

Announcements

- Fruit Ninja late deadline tonight!
  - as always, at least submit something for partial credit by midnight
  - Fruit Ninja Code Debriefs will happen in the following weeks
    - **In total, they are worth 8% of your final grade**
- Cartoon released!
  - early handin: Thursday 10/19
  - on-time handin: Saturday 10/21
  - late handin: Monday 10/23
  - you must complete the Collab Policy Phase 2 quiz, or your project will not be graded
- Cartoon check-ins in Conceptual Hours!
  - be sure to complete the mini-assignment ahead of time, which includes doing the first part of the code!

Socially Responsible Computing

Blockchain & Cryptocurrency II

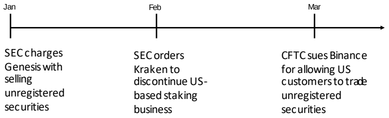
CS15 Fall 2023

From last time, when we discussed FTX…

Caroline Ellison Says She and Sam Bankman-Fried Lied for Years

In her second day testifying at the FTX founder's trial, Ms. Ellison said she had misled lenders and circulated phony financial documents at Mr. Bankman-Fried's request.

Crypto Regulation (2023)



SEC charges Genesis with selling unregistered securities

SEC orders Kraken to discontinue US-based staking business

CFTC sues Binance for allowing US customers to trade unregistered securities

Future of Money

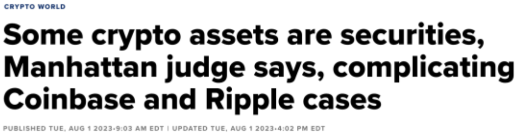
US sues Binance and founder Zhao over 'web of deception'

By **Hannah Lang, Jonathan Stempel** and **Tom Wilson**

June 6, 2023 12:54 AM EDT · Updated 4 months ago

Sources: Forbes, Reuters (2023)

Crypto Regulation (2023)



CRYPTO WORLD

Some crypto assets are securities, Manhattan judge says, complicating Coinbase and Ripple cases

PUBLISHED TUE, AUG 1 2023·9:03 AM EDT | UPDATED TUE, AUG 1 2023·4:02 PM EDT

Source: CNBC (2023)

Scale Issues

Trust in blockchain is reinforced by verifying information across computers

Can lead to blockchains being overwhelmed by the volume of work

BTC was unable to handle more than 7 transactions per second

Environmental Implications

0.4 - 0.9%

of global electricity consumption came from crypto-assets (from 2018 – 2022)

In line with the consumption of the state of Washington

(before the merge) one ETH transaction equaled the power consumption of the average US household over 9 days

Source: White House, Harvard Business Review

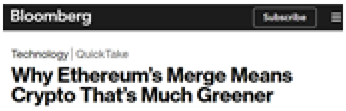
Image sources: Creator - Milos Subasic | Credit - GettyImages

CNBC

TECH

How ethereum's merge made crypto mining more sustainable

PUBLISHED SAT, OCT 22 2022·12:00 PM EDT
UPDATED MON, OCT 24 2022·4:55 AM EDT

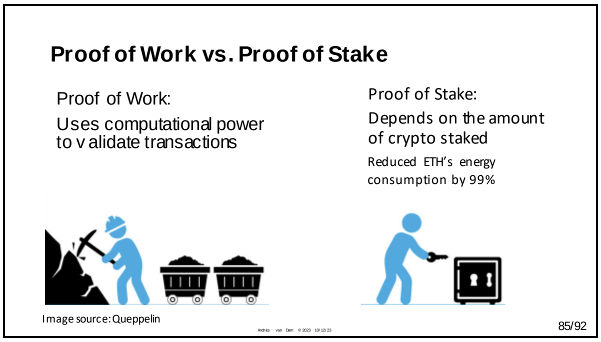


Bloomberg

Technology | QuickTake

Why Ethereum's Merge Means Crypto That's Much Greener

Sources: CNBC, Bloomberg (2022)



Proof of Work vs. Proof of Stake

Proof of Work:

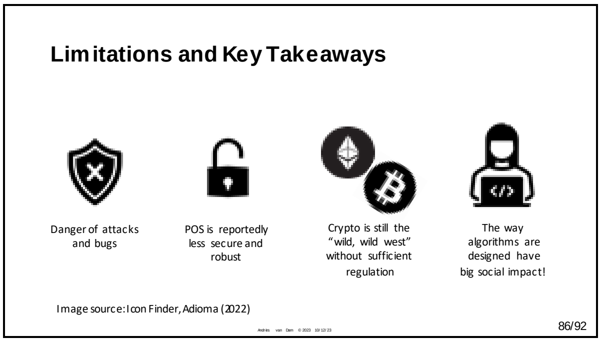
Uses computational power to v alidate transactions

Proof of Stake:

Depends on the amount of crypto staked

Reduced ETH's energy consumption by 99%

Image source: Queppelin



Lim itations and Key Takeaways

Danger of attacks and bugs

POS is reportedly less secure and robust

Crypto is still the "wild, wild west" without sufficient regulation

The way algorithms are designed have big social impact!

Image source: I con Finder, Adioma (2022)