Have questions/concerns about mental/physical health, accessibility, or disability rights within the department?

Stop by Health and Wellness Hours!

Monday 5:00 - 6:00 PM (Anna) in CIT 102
Tuesday 5:00 - 6:00 PM (David) in CIT 102
Wednesday 12:00 - 1:00 PM (Laura) in CIT 221
Friday 5:00 - 6:00 PM (Cristian) in CIT 102
Lecture 11
Graphics Part III – Building up to Cartoon
Review: EventHandlers

- For JavaFX to respond to external stimuli (aka triggers, aka Events), must register an EventHandler with JavaFX so it knows how to respond

- 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)

- If you’ve registered an EventHandler for that trigger, JavaFX will send the Event to the handler and execute the code you registered
# Review: Types of `javafx.event.Events`

<table>
<thead>
<tr>
<th>Trigger</th>
<th>when a button is pressed</th>
<th>when a Timeline's KeyFrame ends a cycle</th>
<th>and many many more! Find them by reading the Javadocs…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of Event</td>
<td><code>ActionEvent</code></td>
<td><code>ActionEvent</code></td>
<td></td>
</tr>
<tr>
<td>Method to register handler</td>
<td><code>setOnAction</code></td>
<td>can only be registered when creating the KeyFrame</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td><code>button.setOnAction((ActionEvent e) -&gt; &lt;method call&gt;);</code></td>
<td>see Graphics II lecture for Timeline examples</td>
<td></td>
</tr>
</tbody>
</table>
Mouse EventHandler 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 `EventHandler` 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 get’ers to access it
  - that bundle of data includes the X location of the click, which we can retrieve using the `getX` and `getY` method

```java
myCircle.setOnMouseClicked((MouseEvent e) ->
    System.out.println(e.getX() + " , " + e.getY());
```
# MouseEvents

<table>
<thead>
<tr>
<th>Trigger</th>
<th>when a mouse is clicked (pressed down, then released)</th>
<th>when a mouse is pressed (not released)</th>
<th>when a mouse is released</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of Event</td>
<td>MouseEvent</td>
<td>MouseEvent</td>
<td>MouseEvent</td>
</tr>
<tr>
<td>Method to register handler</td>
<td>setOnMouseClicked</td>
<td>setOnMousePressed</td>
<td>setOnMouseReleased</td>
</tr>
<tr>
<td>Example</td>
<td>node.setOnMouseClicked((MouseEvent e) -&gt; &lt;method call&gt;);</td>
<td>node.setOnMousePressed((MouseEvent e) -&gt; &lt;method call&gt;);</td>
<td>node.setOnMouseReleased((MouseEvent e) -&gt; &lt;method call&gt;);</td>
</tr>
</tbody>
</table>
## KeyEvents

<table>
<thead>
<tr>
<th>Trigger</th>
<th>when a key is typed (pressed down, then released)</th>
<th>when a key is pressed (not released)</th>
<th>when a key is released</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type of Event</strong></td>
<td>KeyEvent</td>
<td>KeyEvent</td>
<td>KeyEvent</td>
</tr>
<tr>
<td><strong>Method to register handler</strong></td>
<td>setOnKeyTyped</td>
<td>setOnKeyPressed</td>
<td>setOnKeyReleased</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>node.setOnKeyTyped((KeyEvent e) -&gt; method call);</td>
<td>node.setOnKeyPressed((KeyEvent e) -&gt; method call);</td>
<td>node.setOnKeyReleased((KeyEvent e) -&gt; method call);</td>
</tr>
</tbody>
</table>
Outline

• **Example: MovingShape**
• **Constants**
• **Composite Shapes**
  o 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`

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 `EventHandler`s that handle `Buttons`’ `ActionEvents` (clicks) by moving `Shape` correspondingly, within the `ShapeMover` class
1a. Instantiate a PaneOrganizer and store it in the local variable organizer

```java
public class App extends Application {
    @Override
    public void start(Stage stage) {
        PaneOrganizer organizer = new PaneOrganizer();
    }
}
```
MovingShapeApp: App Class (2/3)

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)

```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();
    }
}
```
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!

```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("MovingShape");
        stage.show();
    }
}
```
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 Panes 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-Panes

5. Register Buttons with EventHandlers that handle Buttons’ ActionEvents (clicks) by moving Shape correspondingly, within the ShapeMover class
2a. Instantiate the root Pane and store it in the instance variable root

```java
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

```java
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 13).

```java
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 `EventHandlers` 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 **EventHandler** and link it to the buttons

3a. Make the constructor of **ShapeMover** take in the root **Pane** (association between container and component, created in **PaneOrganizer**, see slide 11)

```java
public class ShapeMover {
    private Ellipse _ellipse;
    public ShapeMover() {
        _ellipse = ellipse;
        this.setupShape();
        this.setupButtons();
    }
}
```
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

```java
public class ShapeMover {
    private Ellipse ellipse;

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

```java
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 `ShapeMover` 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 EventHandlers 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
  o setupShape() fills and positions Ellipse
  o setupButtons() adds and positions Buttons, and registers them with their appropriate EventHandlers

• Helper methods of the form setupX() are fancy initializing assignments. Should be used to initialize variables, but not for arbitrary/non-initializing code.

• Generally, helper methods should be private – more on this in a moment
MovingShapeApp: setupShape() helper method

• For this application, “helper method” setupShape() will only set fill color and position Ellipse in Pane using absolute positioning

• Helper method is private—why is this good practice?
  o only ShapeMover class should be allowed to initialize the color and location of the Ellipse
  o private methods, like private instance variables, are only pseudo-inherited and are therefore not accessible to subclasses—though inherited superclass methods may make use of them w/o the subclass knowing about them!

```java
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);
    }
}
```
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!

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

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!
Aside: **BorderPane** Class (3/3)

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

4a. Change root to a BorderPane

```java
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

4a. Change root to a BorderPane

4b. Create a Pane to contain Ellipse. To add shapePane to center of BorderPane, call setCenter(shapePane) on root

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

```java
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

```java
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)

- and **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

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

```java
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");
    }
}
```
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(…)`

```java
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);
    }
}
```
MovingShapeApp: setupButtons() method (3/4)

4h. Set horizontal spacing between Buttons as you like

```java
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);
    }
}
```
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 EventHandlers, but first we should define the EventHandler.

```java
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 setupButtons(HBox buttonPane) {
        Button b1 = new Button("move left");
        Button b2 = new Button("move right");
        buttonPane.getChildren().add(b1);
        buttonPane.getChildren().add(b2);
        buttonPane.setSpacing(30);
    }

    // setupShape elided
}
```
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 EventHandlers that handle Buttons’ ActionEvents (clicks) by moving Shape correspondingly, within the ShapeMover class**
Aside: Creating **EventHandler**s

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

```java
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: MoveHandler (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

```java
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;
        // other code elided
    }
}
```
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

5c. Move the ellipse’s x-location to `newXLoc`
Register Buttons with their EventHandlers 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));
    }
}

This is where we set xChange
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, Buttons, and an EventHandler (MoveHandler)
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 PanegetRoot() {
        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);
    }
}
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

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

The Real 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("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;
    }
}
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)
  o 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);

        this.setXLoc(Constants.START_X_OFFSET);
    }

    public void setXLoc(double x) {
        this.face.setCenterX(x - Constants.EYE_X / 2);
    }

    public double getXLoc() {
        return this.face.getCenterX();
    }
}

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 incorrect!
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:

```java
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?
  o move left
  o 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
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
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!
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(\(\text{Constants.X, Constants.Y}\));
    this.window = new Rectangle(\(\text{Constants.WIND_X, Constants.WIND_Y}\));
    this.door = new Rectangle(\(\text{Constants.DOOR_X, Constants.DOOR_Y}\));
    //code to fill foundation, window, door elided
    housePane.getChildren().addAll(this.foundation, this.window, this.door);
    this.setXLoc(\(\text{Constants.INITIAL_X_OFFSET}\));
} `

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

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(\(\text{Constants.INITIAL_X_OFFSET}\));
} `

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

```java
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, contains an Alien called alien
- Change shapePane to be an alienPane (we could have called it anything!)

```java
public class AlienMover {
    private Alien alien;
    public ShapeMover(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(Defaults.X_OFFSET);
        this.ellipse.setCenterY(Defaults.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(Defaults.BUTTON_SPACING);
        b1.setOnAction((ActionEvent e) -> this.moveEllipse(-1 * Defaults.DISTANCE_X));
        b2.setOnAction((ActionEvent e) -> this.moveEllipse(Defaults.DISTANCE_X));
    }
}
```

// moveEllipse elided
• setupShape() method is no longer needed, as we now setup the Alien within the Alien class
  o remember that we set a default location for the Alien in its constructor:
  
this.setXLoc(Variables.START_X_OFFSET);
MovingAlien: AlienMover Class (3/3)

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

```java
public class AlienMover {
    private Alien alien;
    public ShapeMover(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);
    }
}
```
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

```java
public class AlienMover {
    private Alien alien;
    public ShapeMover(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());
    }
}

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 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 Alien {
    private Ellipse face;
    private Ellipse leftEye;
    private Ellipse rightEye;
    public Alien(Pane root) {
        this.face = new Ellipse(0, 0);
        this.face.setFill(Color.CHARTREUSE);
        this.leftEye = new Ellipse(0, 0);
        this.rightEye = new Ellipse(0, 0);
        this.setXLoc(0);
        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);
    }
}

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.moveRight());
    }
    public Pane getRoot() {
        return this.root;
    }
}
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.
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 the **KeyFrame**, your cartoon should move (or change in other ways) a small amount!
  - reminder: if we move fast enough and in small enough increments, we get smooth motion!
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 Sliders, Spinners, and ColorPickers
  - 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

- Cartoon released!
  - early handin: Thursday 10/21
  - on-time handin: Saturday 10/23
  - late handin: Monday 10/25
  - you must complete the Collab Policy Phase 2 quiz, or your project will not be graded

- Cartoon check-ins this Saturday – Monday
  - if you haven’t heard from your mentor TA yet, reach out to the HTAs
  - be sure to complete the mini-assignment ahead of time, which includes doing the first part of the code!

- Please fill out this form with feedback about Pong and Tic Tac Toe!
Topics in Socially-Responsible Computing

Privacy & Surveillance II
**Case Study: Apple + San Bernardino**

- FBI sought to unlock iPhone of a San Bernardino shooter (Dec. 2015)
  - Requested the Court order Apple to provide assistance in decrypting the code
  - Tech didn’t exist! → requested a custom operating system to disable key security features
- Apple opposed
  - Claimed it was unlawful and unconstitutional
  - Undermines security of all Apple devices
  - Sets a dangerous precedent for future cases

Photo: VOX
Case Study: Apple + San Bernardino (Cont’d)

• Hearing postponed after FBI found a third-party to unlock the company
  • Small Australian hacking firm Azimuth Security in 2016
  • Identity of company remained secret for ~5 years

• "The government could extend this breach of privacy and demand that Apple build surveillance software to intercept your messages, access your health records or financial data, track your location, or even access your phone’s microphone or camera without your knowledge.” – Tim Cook

Photo: HackRead
Case Study: Apple CSAM

• Concern about spread of Child Sexual Abuse Material (CSAM)
  • If user searches for topics related to CSAM, Apple will direct towards recourses for reporting it / getting help
  • Parental control option to Messages: obscure sexually explicit pictures for users under 18 and sends alert if user under 12 views/sends
    • Scans for nudity on-device to protect privacy
  • Develop technology to detect CSAM: NeuralHash
    • Designed to identify known CSAM on iCloud photos and report to Apple moderators
    • Device locally check pictures against known CSAM—if enough matches are detected, automatically will alert Apple moderators
    • If moderator confirms, the account will be disabled and images will be reported to legal authorities
Case Study: Apple CSAM

- CSAM scanning used by Facebook, Twitter, Reddit and other companies already use
  - Typically runs remotely and looks at files stored on a server
- Apple’s system would check locally
  - NeuralHash breaks pictures into hashes (strings of numbers) and compares against known CSAM hashes
  - If a certain number of matches are found, photos are decrypted and flagged to moderators

Photo: Datanami
Case Study: Apple CSAM

• Received backlash
  • Create surveillance systems that work directly on phone or tablet
  • Could provide blueprint for breaking secure end-to-end encryption
    • End-to-end encryption makes data unreadable for anyone besides sender / receiver (iMessages)
  • Client-side scanning: analyzing files or messages in an app before they’re sent in encrypted form
  • Capability to reach into user’s phone sets dangerous precedent

• Edward Snowden: “Apple’s new system, regardless of how anyone tries to justify it, will permanently redefine what belongs to you, and what belongs to them”
Case Study: Apple CSAM

• On September 3rd, 2021 decided to delay release

Update as of September 3, 2021: Previously we announced plans for features intended to help protect children from predators who use communication tools to recruit and exploit them and to help limit the spread of Child Sexual Abuse Material. Based on feedback from customers, advocacy groups, researchers, and others, we have decided to take additional time over the coming months to collect input and make improvements before releasing these critically important child safety features.

Announcement by Apple
Case Study: Apple CSAM

• How does the right to privacy fit into other rights?
  • Security vs. privacy

• Should elicit activities be protected?
  • Laundering money, child pornography, trafficking all protected under absolute privacy laws

• Raises tough philosophical questions we may not have the answers to
Privacy rocks! More you might like:

- [Apple vs. FBI](https://www.cnbc.com/2016/12/08/apple-vs-fbi.html) – CNBC
- [The FBI wanted to unlock the San Bernardino shooter’s iPhone. It turned to a little-known Australian firm](https://www.washingtonpost.com/national/crime/2016/12/08/apple-raises-concerns-about-fbi-plan-to-unlock-iphone/?utm_term=0120161208106116) — Washington Post
- [Apple vs. FBI](https://www.epic.org/2016/12/08/apple-raises-concerns-about-fbi-plan-to-unlock-iphone.html) — epic.org
- [Apple’s controversial new child protection features, explained](https://www.theverge.com/2016/12/8/13413020/apple-controversial-child-protection-features) — The Verge
- [Snowden Slams Apple CSAM](https://www.forbes.com/sites/ericparé/2021/01/10/snowden-slamms-apple-csam/?sh=3e3f800652e9) — Forbes
- [Apple delays controversial child protection features after privacy outcry](https://www.theverge.com/2021/1/10/22140654/apple-csam-delay-privacy-crisis) — The Verge