Computer Programming – CS 6011
Lecture 9: Creating Widget Class

Fall 2023

Topics

- Destroying an object in Java
- In-class example of creating an Audio Component Widget
  - Moving a widget
  - Connecting the widget to the speaker

Object destruction…

- How to destroy an object in C++?
  - delete
- How to destroy an object in Java?
  - `Car c = new Car();`
  - Forget about it / let it go out of scope / set it to null…
  - `c = null;`
    - Variable `c` still exists, but the object c was pointing at (referring to) is automatically "cleaned up" / deleted by the Java garbage collector.
- How to destroy a (Javafx) Widget?
  - Make sure no one knows about it… Let's see this in more detail.

Avoiding memory leaks in Java?

- But first let's get a widget to destroy…

```
SineWaveWidget sw = new SineWaveWidget( 440 );
```

- We need to keep track of this (all?) widget…

```
ArrayList<AudioWidgets> components_ = new ArrayList<>();
components_.add( sw );
```

Will `sw` show up on our stage now?

No, we must also give the component (widget) to javafx to display for us:

```
root.getChildren().add( sw );
```

- Now to delete sw… (Perhaps the user pressed the 'x' (close) button on this widget.)

```
root.getChildren().remove( sw ); // sw no longer is displayed…
```

Done?

No! Need to make sure that *everyone* forgets about `sw` so that Java will get rid of it for us…

```
components_.remove( sw );
```

Playing the Clip

- What is this code doing?

```
1:  Clip c = AudioSystem.getClip(); // Not our AudioClip class
2:  byte[] data = mySineWave.getClip().getData();
3:  // Reads data from our byte array to play it.
4:  c.open( format16, data, 0, data.length );
5:  c.start(); // Actually starts playing the sound.
6:  // Don't end the program until the sound finishes playing…
7:  while( c.getFramePosition() < AudioClip.TOTAL_SAMPLES || c.isActive() ||
c.isRunning() ){
8:      // Do nothing.
9:  }
10: c.close();
```

- Try this with a GUI and what will happen?
  - The GUI will lock-up until the while loop finishes…

Clip Listener

```
AudioFormat format16 = new AudioFormat(44100, 16, 1, true, false );
Clip c = AudioSystem.getClip(); // Not our AudioClip class
AudioListener listener = new AudioListener( c );

byte[] data = ac.getClip().getData();
// Reads data from our byte array to play it.
c.open( format16, data, 0, data.length );
c.start(); // Actually starts playing the sound.
// Handle taking care of the sound when it ends…
c.addLineListener( listener ); // Replaces while loop
// c.close(); should be removed as this is done in the AudioListener class
```

The Listener Class

```
class AudioListener implements LineListener {
   public AudioListener( Clip c ) {
      clip_ = c;
   }
   @Override
   public void update(LineEvent event) {
      if( event.getType() == LineEvent.Type.STOP ) {
         // System.out.println("close clip");
         clip_.close();
      }
   }
   private Clip clip_;
}
```

Line Listener Using Lambda

- This is a relatively new feature of in Java and makes the amount of code to handle an event much smaller.

```
c.addLineListener( e->handleAudio( e, c ) );

// And define the handleAudio() Function
private void handleAudio( LineEvent e, Clip c ) {
    if( e.getType() == LineEvent.Type.STOP ) {
        c.close();
      }
}
```

Constructing an AC Widget

```
public class AudioComponentWidget extends Pane {
        AudioComponentWidget(  AudioComponent ac, AnchorPane parent, String componentName ) {
                …
    // this is a Widget (it is a “Pane”), and thus we need to add the baseLayout_ to “ourself”.
                this.getChildren().add( baseLayout_ );
     // The parent_ (passed in when this was created) is the AnchorPane that was put into the
     // (very) top level BorderPane’s center.  By adding this to it, this Audio Component Widget
     // will be drawn on the screen:
                parent_.getChildren().add( this ); // <− add ourself to our parent
```

Making a Line Move

```
private void connectionLineDragHandler( MouseEvent e ) {
                    // Because this widget is in the center pane of the BorderPane
                    // (grand) parent, we need to take into account its offset.
                    // In other words, the point (0, 0) in scene coordinates is at the
                    // very top left of the application scene.  However, (0, 0) with respect to the line's
                    // coordinates is offset by the size of the "top" pane (in the BorderPane)
          Bounds offset = parent_.getBoundsInParent();
          line_.setEndX( e.getSceneX() - offset.getMinX() );
          line_.setEndY( e.getSceneY() - offset.getMinY() );
}
```

Synthesizer App

- The synthesizer app needs to keep track of the number of connections to the speaker.
- Then when the user presses “Play”, it loops through those connections and plays each of them.

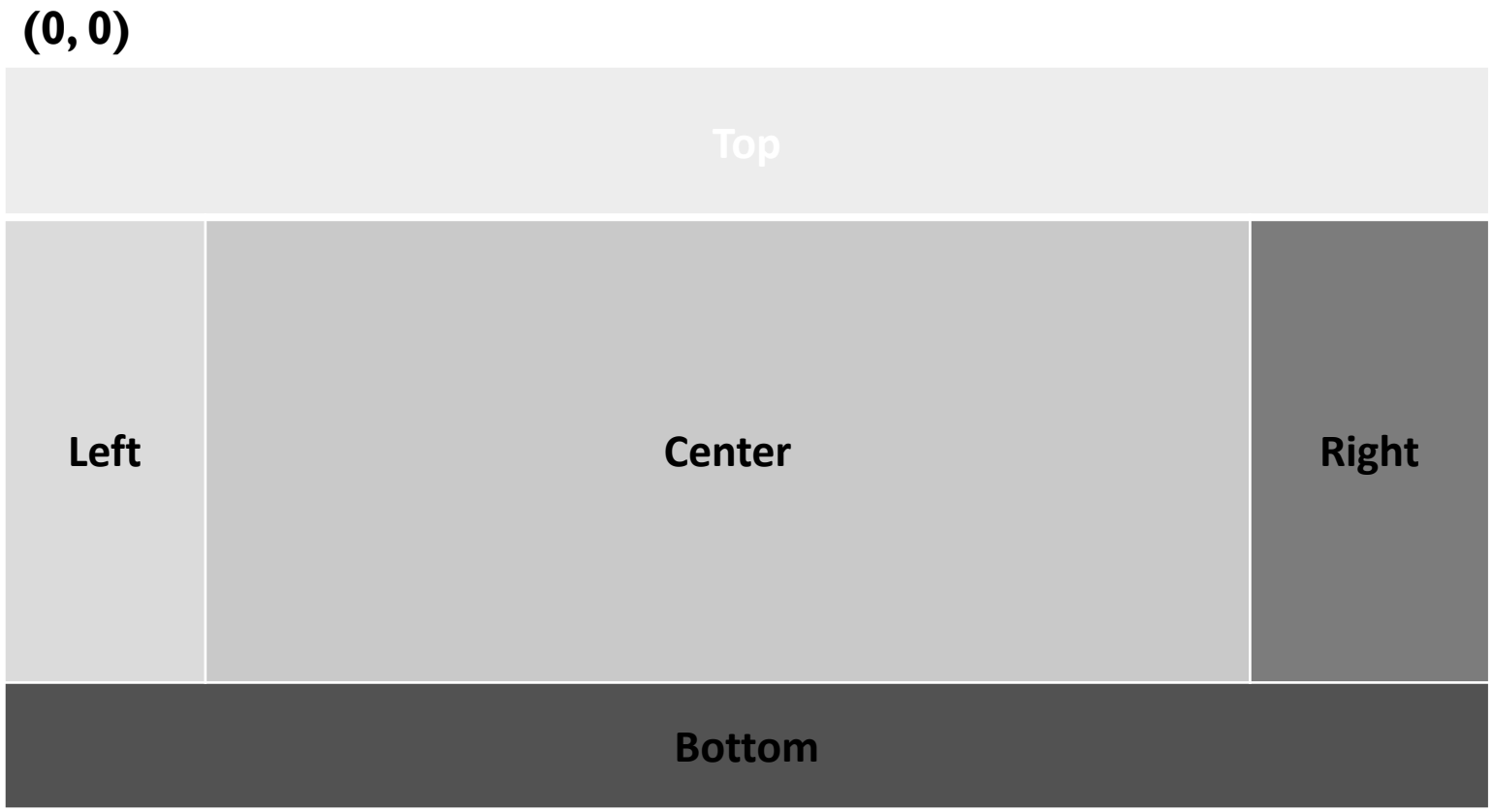
```
public static ArrayList<AudioComponent> speakerConnections_ = new ArrayList<>();
```

- When an Audio Component Widget is deleted (closed) or its connection to the speaker is removed, the `speakerConnections_` should be updated to reflect this.

Widget Design

- `BorderPane root`

- What do we place in the center of `root`?
  - `root.setCenter( ? )`



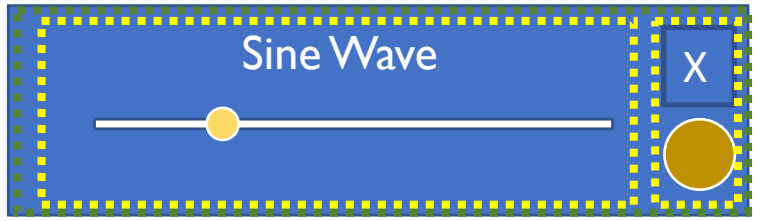
Widget Design

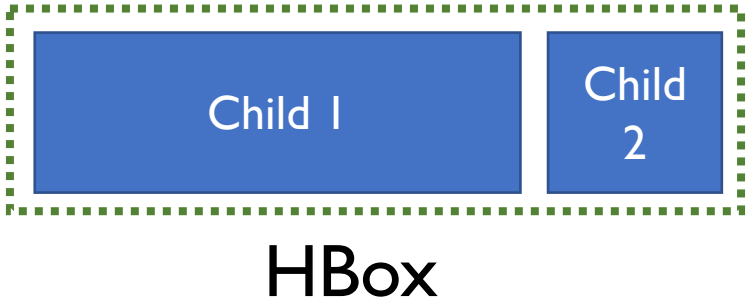
- `AnchorPane centerPanel`

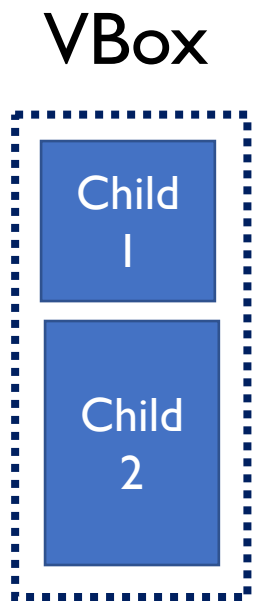
- And what do we place in the centerPanel?
  - `AudioComponentWidget`

AudioComponentWidget

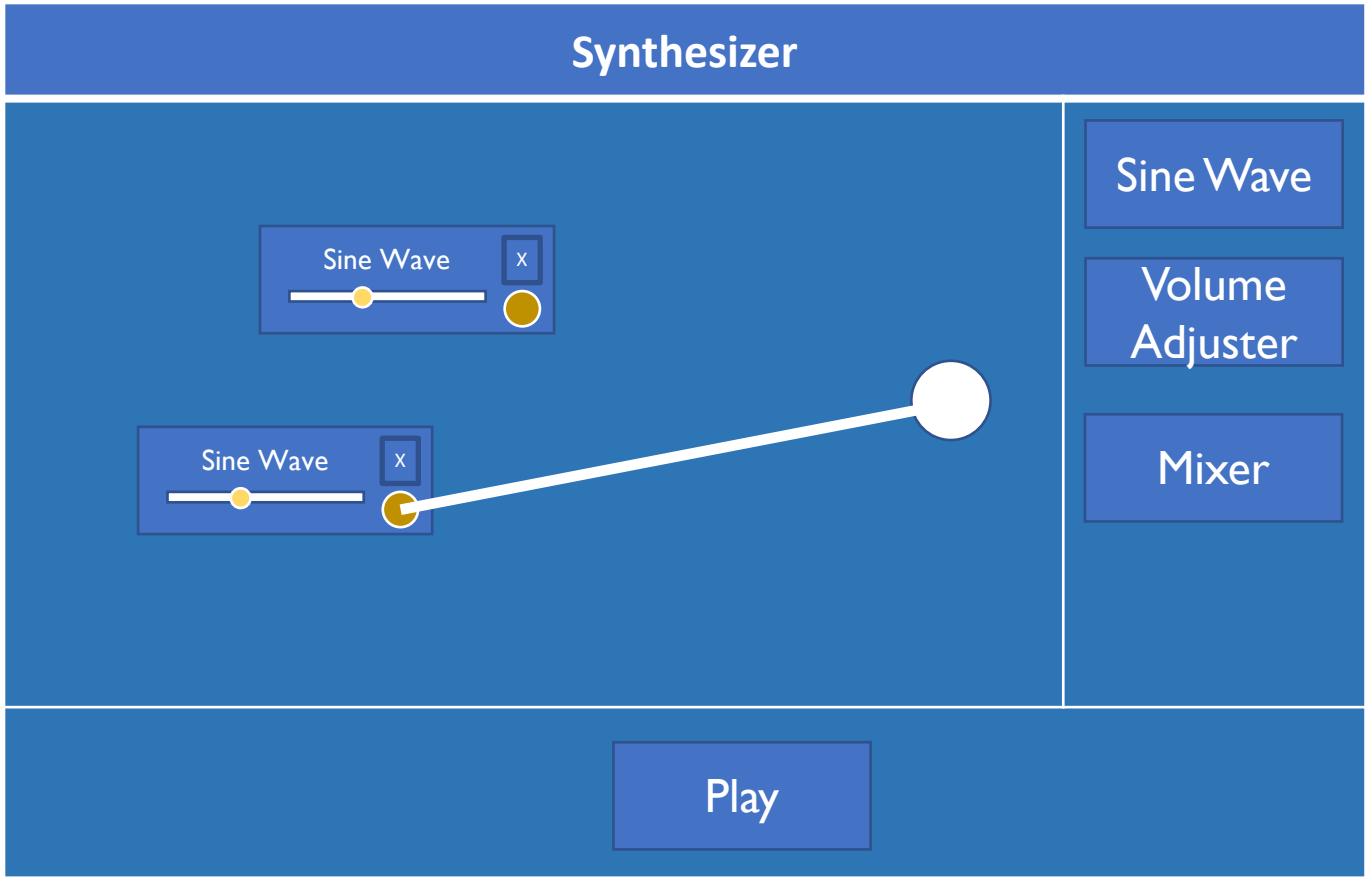
- How to lay all these pieces out?
  - Need to “Group” them…
  - Different ways…
  - HBoxes
  - VBoxes
- What is the base layout (container / pane) for the ACW?
  - Lots of choices…
  - Perhaps a HBox()





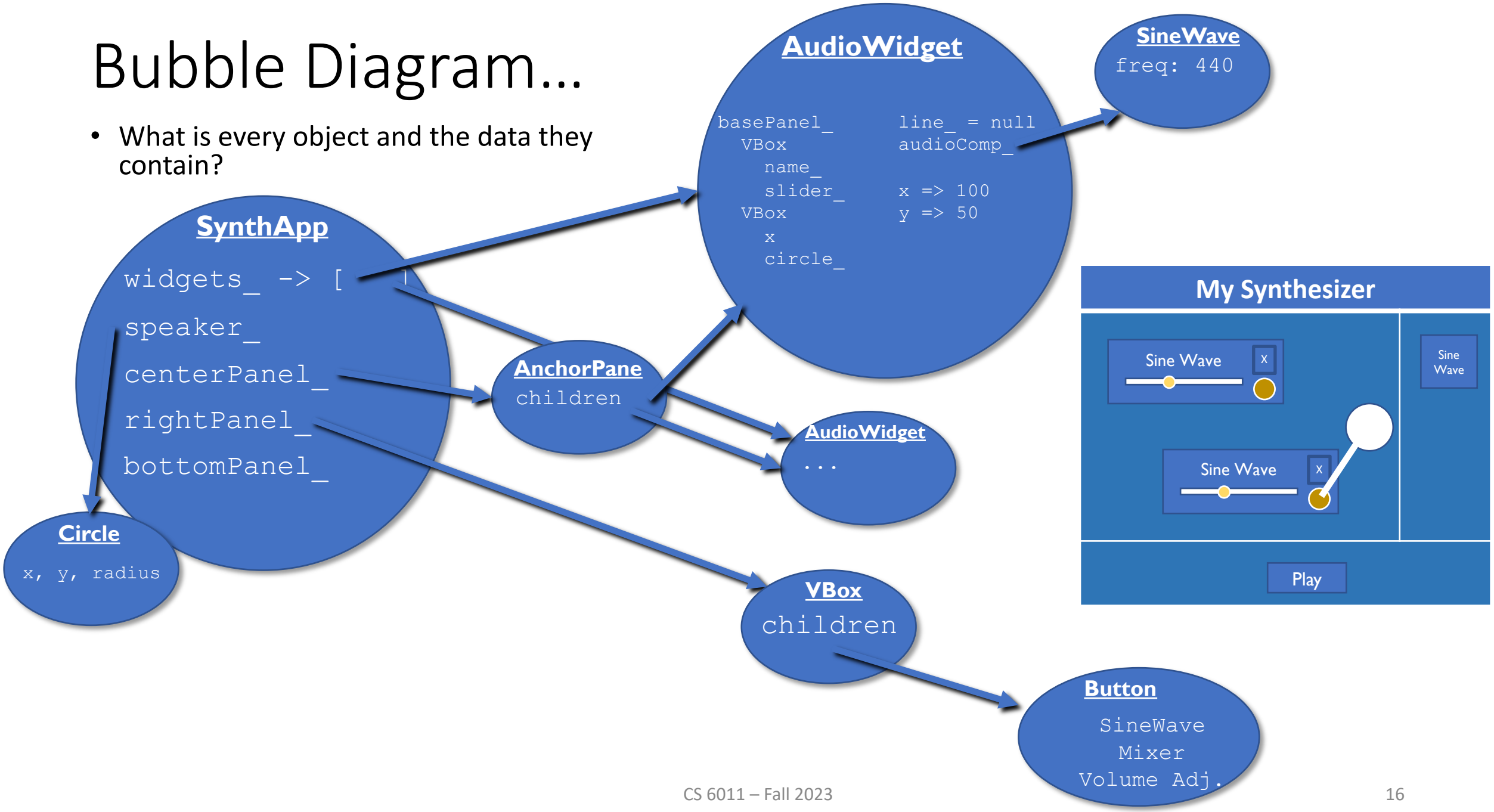


Synthesizer GUI



Bubble Diagram…

- What is every object and the data they contain?



Thursday Assignments

- Code Reviews:
  - Synthesizer Part 1 (4a) and Synthesizer GUI Part 1 (4b)
- Continue to work on your Synthesizer today and tomorrow.