Coding Workshop: refactoring the GUI into a class

Introduction

In this worksheet, we will refactor the GUI code into its own class, called DeckGUI. Then we can instantiate a complete set of GUI controls for a player with one line of code. We can also delegate control of the DJAudioPlayer to the new class.

In an earlier worksheet, you may have tried adding two audio players (or 'decks' in DJ-speak) to your application. You will have noticed that you needed to duplicate all the GUI component code. You might have needed names for the different buttons like 'playButton1' and 'playButton2'. Imagine that you wanted to add a third player - things would get really messy.

Now we are going to make things neater by wrapping up all that GUI code in a class. The class will be a Component, so we can use it just like any other JUCE Component, e.g. call addAndMAkeVisible, setBounds and so on. We will also pass it a DJAudioPlayer which it can control.

Set up a new DeckGUI class

- Load up your project in Projucer
- Open up the File Explorer Tab
- Right-click on the Source folder and select Add new Component split between CPP and header file, like this:

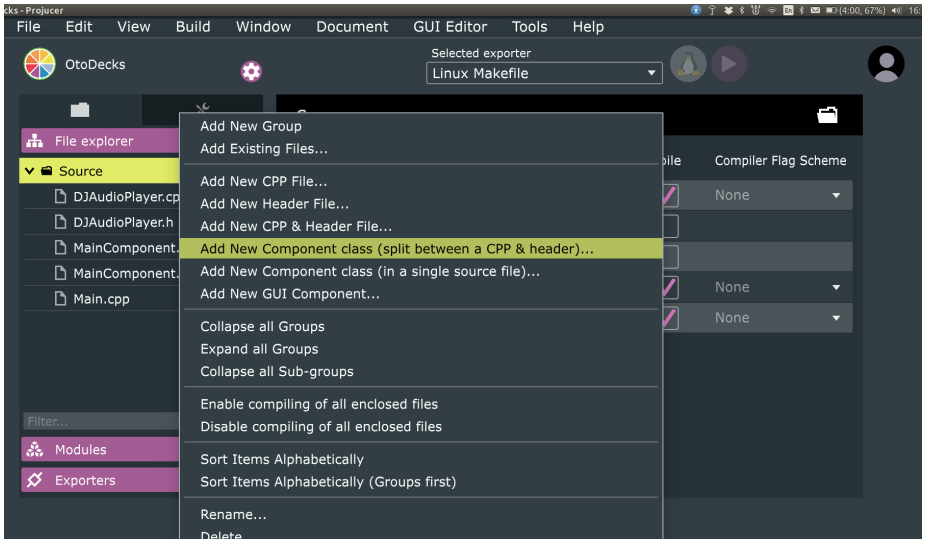

Figure 1: Freshly minted Component

- Call the file DeckGUI
- It should create the new files in your Source folder
- Very important - save the project out of Projucer then re-open it in your IDE. If you do not save it, the new file will not appear in the build.

Test out the basic Component

Now try adding the Component to your MainComponent. It can be treated just like any other component. Here are instructions, but see if you can figure out how to do it yourself first.

Include the header file from MainComponent.h:

```
#include "DeckGUI.h"
```

Add a field to the private section of your MainComponent.h:

```
DeckGUI deck1;
```

Call addAndMakeVisible in the constructor in MainComponent.cpp:

```
addAndMakeVisible(deck1);
```

Call setBounds in the resized function in MainComponent.cpp:

```
deck1.setBounds(0, getHeight()/5 * 4, getWidth(), getHeight()/5);
```

You might need to adjust the parameters on the setBounds function depending on how you have your GUI laid out.

You should be able to see the default graphics for the DeckGUI component in your main application.

Customise the Component

Now we are going to customise the Component, so it has the controls we need. We'll add play, load and stop buttons. We'll add a position slider and a volume slider.

Add the button and slider data members

In DeckGUI.h, private section:

```
TextButton playButton;
TextButton stopButton;
TextButton loadButton;
Slider volumeSlider;
Slider positionSlider;
Slider speedSlider;
FileChooser fChooser{"Select a file..."};
```

Implement DeckGUI's constructor and resized functions

In DeckGUI.cpp's constructor (yes we can add Components to our DeckGUI Component, just as we did to the MainComponent)

```
DeckGUI::DeckGUI()
{
    addAndMakeVisible(playButton);
    addAndMakeVisible(stopButton);
    addAndMakeVisible(loadButton);
    addAndMakeVisible(volumeSlider);
    addAndMakeVisible(positionSlider);
    addAndMakeVisible(speedSlider);

}
```

In DeckGUI's resized method:

```
void DeckGUI::resized()
{
    float rowH = getWidth() / 6;
    playButton.setBounds(0, 0, getWidth(), rowH);
    stopButton.setBounds(0, rowH, getWidth(), rowH);
    volumeSlider.setBounds(0, rowH*2, getWidth(), rowH);
    positionSlider.setBounds(0, rowH*3, getWidth(), rowH);
    speedSlider.setBounds(0, rowH*4, getWidth(), rowH);
    loadButton.setBounds(0, rowH*5, getWidth(), rowH);
}
```

Clear out the paint function to this:

```
void DeckGUI::paint (Graphics& g)
{
    g.fillAll (getLookAndFeel().findColour (ResizableWindow::backgroundColourId));   // clea
}
```

Test it

Now build and rerun the program. If you have left in all the buttons on the MainComponent class, you will see something a bit confusing, like this:

The DeckGUI has been squished into a space below the other MainComponent GUI components. Notice that the DeckGUI has scaled itself into the available area.. This is because getWidth and getHeight will tell you the width and height available to that Component itself, not the whole window. If you set the size of the subcomponents (the buttons and sliders inside DeckGUI) relative to the width and height, it will scale automatically.

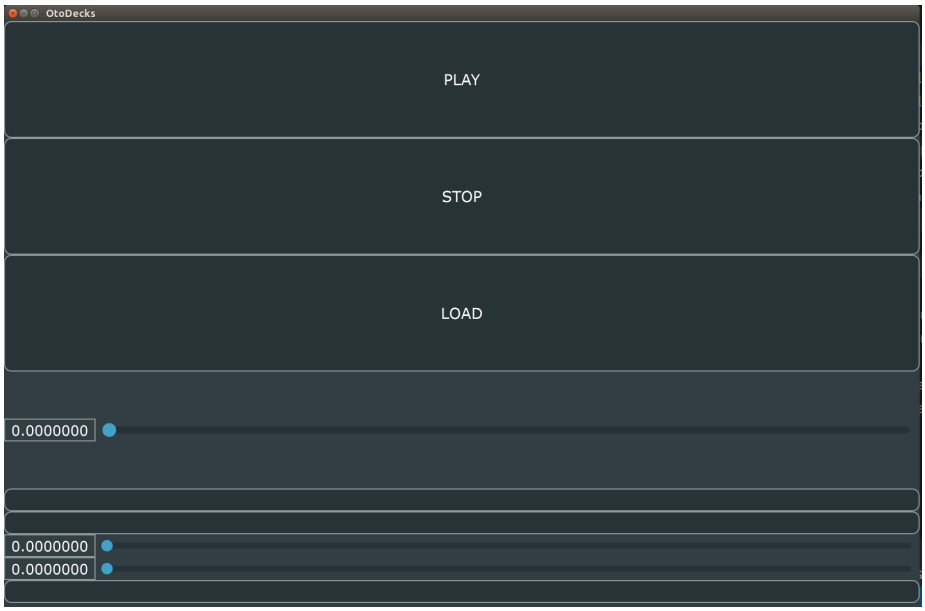

Figure 2: Tiny DeckGUI

Get rid of the old components on MainComponent

In MainComponent.cpp (not DeckGUI.cpp!) comment out the calls to addAndMakeVisible on all but the DeckGUI component:

Change MainComponent::resized to this:

```
void MainComponent::resized()
{
    deck1.setBounds(0, 0, getWidth()/2, getHeight());
}
```

Note that I set it to be as high as the whole window but half as wide. Compile and run, and you should see something like this:

Go ahead and add some labels to your buttons and sliders, then set the range of your sliders to between 0 and 1. You might want to customise the layout as well.

Add a DJAudioPlayer to control

Clicking on the buttons does not do anything. This is because we have not implemented any event listening code in our DeckGUI component. We'll do that in a minute, but first we need a DJAudioPlayer to control.

Since DeckGUI will always interact with a DJAudioPlayer, we'll force the person creating the object to pass in a DJAudioPlayer.

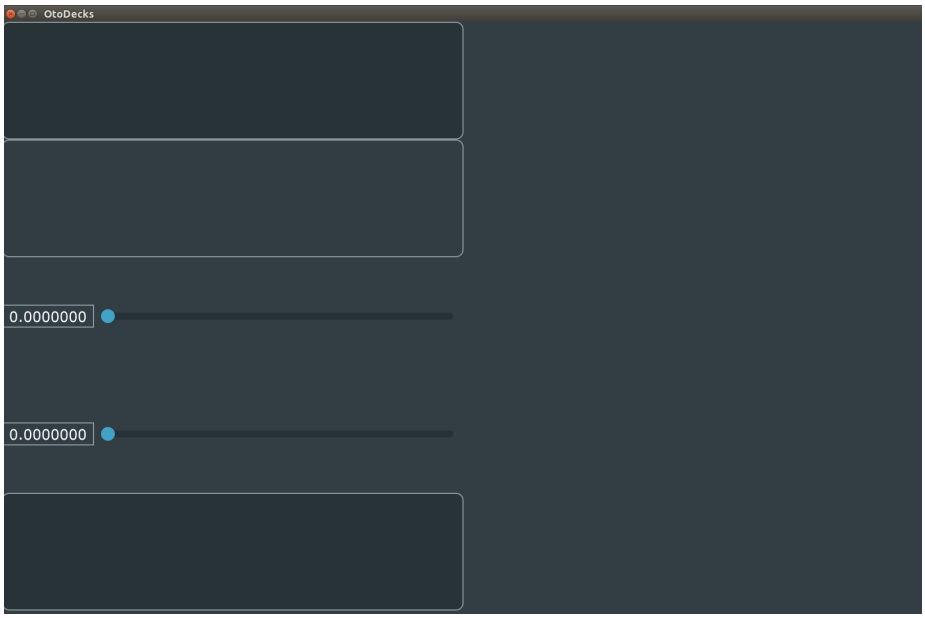

Figure 3: DeckGUI taking up half the window

In DeckGUI.h, include DJAudioPlayer.h:

```
#include "DJAudioPlayer.h"
```

Change the constructor signature to this:

```
DeckGUI(DJAudioPlayer* _djAudioPlayer);
```

Add a field to the private section:

```
DJAudioPlayer* djAudioPlayerplayer.;
```

In DeckGUI.cpp, deal with this in the constructor:

```
DeckGUI::DeckGUI(DJAudioPlayer* _djAudioPlayer) : djAudioPlayer{_djAudioPlayer}
```

Notice we are using C++11 style member initialisation to receive the player and assign it to the djAudioPlayer data member.

Where does it get the DJAudioPlayer object from? Well, we can pass one through from the MainComponent, in MainComponent.h:

```
DJAudioPlayer player;
DeckGUI deck1{&player};
```

DeckGUI's constructor wants a pointer, which is the memory address of some data. So we use the address of operator '&' to send it the address of the player variable. This is similar to passing data by reference as DeckGUI has access

to the original DJAudioPlayer, not a copy. However, passing things around by pointers is a more common pattern in JUCE. For example, consider the event listeners.

Implement the event listener in DeckGUI

In DeckGUI.h, change the class inheritance relationship to this:

```
class DeckGUI    : public Component,
                   public Button::Listener,
                   public Slider::Listener
```

That forces us to implement pure virtual functions from Button::Listener and Slider::Listener. Add these into the public section of the DeckGUI class in DeckGUI.h:

```
void buttonClicked(Button* button) override;
void sliderValueChanged (Slider *slider) override;
```

In DeckGUI.cpp, in the constructor DeckGUI::DeckGUI, register with the buttons and sliders as a listener:

```
playButton.addListener(this);
stopButton.addListener(this);
volumeSlider.addListener(this);
positionSlider.addListener(this);
speedSlider.addListener(this);

loadButton.addListener(this);
```

In DeckGUI.cpp, implement the event listener functions by hooking them up to the DJAudioPlayer:

```
void DeckGUI::buttonClicked(Button* button)
{
    if (button == &playButton )
    {
        djAudioPlayer->setPosition(0);
        djAudioPlayer->play();
    }
    if (button == &stopButton )
    {
        djAudioPlayer->stop();
    }
    if (button == &loadButton)
    {
        auto fileChooserFlags =
        FileBrowserComponent::canSelectFiles;
        fChooser.launchAsync(
            fileChooserFlags, [this](const FileChooser& chooser)
        {
            auto chosenFile = chooser.getResult();
            player->loadURL(URL{chosenFile});

        });
}
}

void DeckGUI::sliderValueChanged(Slider* slider)
{
    if (slider == &volumeSlider)
    {
        djAudioPlayer->setGain(slider->getValue());
    }
    if (slider == &positionSlider)
    {
        djAudioPlayer->setPositionRelative(slider->getValue());
    }
     if (slider == &speedSlider)
    {
        djAudioPlayer->setSpeed(slider->getValue());
    }
}
```

Compile and test - load a file, play, stop, change the gain, change position.

Add another player

Now we are ready to experience the joy of object-oriented programming. Let's add another deck!

Clean up the MainComponent code

First of all, let's clean up the MainComponent a bit - it has some GUI widgets sitting around that we don't want anymore.

In MainComponent.h, add data members to the private sections (noting I've renamed player to player1):

```
DJAudioPlayer player1;
DeckGUI deck1{&player1};

DJAudioPlayer player2;
DeckGUI deck2{&player2};
```

Go through and update any references to the player variable. Change them to player1.

Remove the GUI components from MainComponent.h. You just need this in private now:

```
private:
    DJAudioPlayer player1;
    DeckGUI deck1{&player1};

    DJAudioPlayer player2;
    DeckGUI deck2{&player2};
```

Clean out the code in MainComponenet's event listeners:

```
void MainComponent::buttonClicked(Button* button)
{
}

void MainComponent::sliderValueChanged(Slider* slider)
{
}
```

If you like, you can altogether remove the inheritance relationship between MainComponent and the event listener classes.

Set up the DeckGUI components:

In the constructor:

```
MainComponent::MainComponent()
{
```

```
    setSize (800, 600);
    addAndMakeVisible(deck1);
    addAndMakeVisible(deck2);

...
```

In resized:

```
void MainComponent::resized()
{
    deck1.setBounds(0, 0, getWidth()/2, getHeight());
    deck2.setBounds(getWidth()/2, 0, getWidth()/2, getHeight());
}
```

Compile and test. You should see something like this:

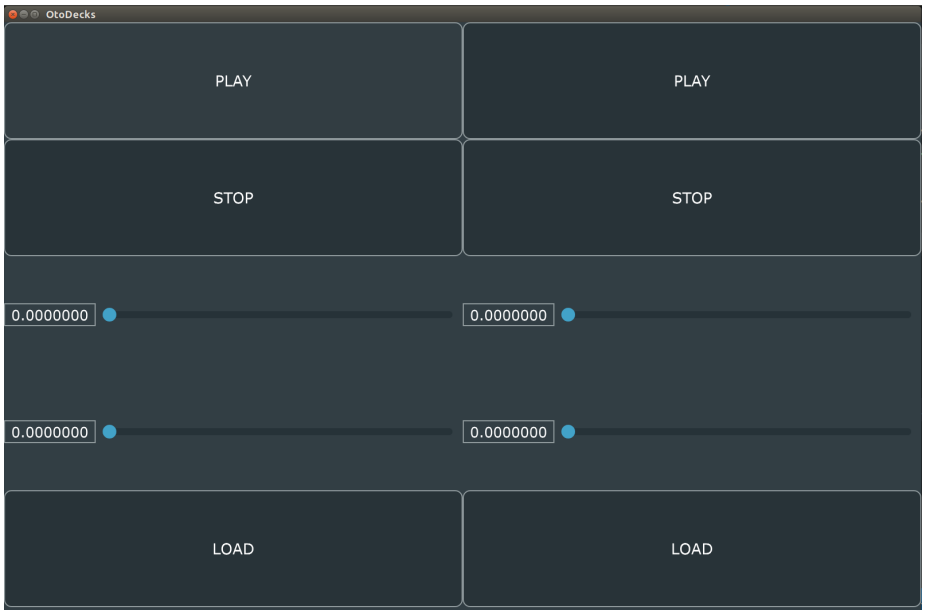

Figure 4: GUI showing two decks

You can see two decks. But deck 2 does not play any sound! Oh no.

How to play two files? Add a mixer!

The final step is to add some more audio code so that we can mix the sound of player1 and player2 together and hear them both at the same time.

In MainComponent.h, in the private section of the class definition:

```
MixerAudioSource mixerSource;
```

In MainComponent.cpp, change the audio setup, play and shutdown code to this:

```
void MainComponent::prepareToPlay (int samplesPerBlockExpected, double sampleRate)
{
    mixerSource.addInputSource(&player1, false);
    mixerSource.addInputSource(&player2, false);
    // note that this will call prepareToPlay
    // automatically on the two players
    mixerSource.prepareToPlay(samplesPerBlockExpected, sampleRate);
    // so this is not needed:
    //player1.prepareToPlay(samplesPerBlockExpected, sampleRate);
    //player2.prepareToPlay(samplesPerBlockExpected, sampleRate);
}

void MainComponent::getNextAudioBlock (const AudioSourceChannelInfo& bufferToFill)
{
    mixerSource.getNextAudioBlock(bufferToFill);
}

void MainComponent::releaseResources()
{
    mixerSource.removeAllInputs();
    mixerSource.releaseResources();
    player1.releaseResources();
    player2.releaseResources();
}
```

Notes:

- in prepareToPlay, we set up the mixer configuration, which is essentially adding input sources. The polymorphism comes in nicely here - the addInputSource just wants any ancestor of AudioSource that has the audio functions on it.
- getNextAudioBlock is now really simple! mixerSource takes control over the players as they are not its inputs and we just ask for audio from the mixer.
- the releaseResources code unplugs the inputs to the mixer, the releases all audio resources.

Test it out. You should now be able to play two tracks at the same time. Cool!

Challenge

Design a different GUI to control the DJAudioPlayer class.

Conclusion

In this worksheet we have refactored the GUI code into a new class. The new class makes it easy to create multiple instances of the DJ deck user interface.

The complete DeckGUI.h

Note there might be slight variations between your version and this version.

```
#pragma once

#include "../JuceLibraryCode/JuceHeader.h"
#include "DJAudioPlayer.h"

//==============================================================================
/*
*/
class DeckGUI    : public Component,
                   public Button::Listener,
                   public Slider::Listener,
                   public FileDragAndDropTarget
{
public:
    DeckGUI(DJAudioPlayer* player);
    ~DeckGUI();

    void paint (Graphics&) override;
    void resized() override;

    /** implement Button::Listener */
    void buttonClicked (Button *) override;

    /** implement Slider::Listener */
    void sliderValueChanged (Slider *slider) override;

    bool isInterestedInFileDrag (const StringArray &files) override;
    void filesDropped (const StringArray &files, int x, int y) override;



private:

    TextButton playButton{"PLAY"};
    TextButton stopButton{"STOP"};
    TextButton loadButton{"LOAD"};

    Slider volSlider;
```

```
    Slider speedSlider;
    Slider posSlider;

    FileChooser fChooser{"Select a file..."};


    DJAudioPlayer* player;

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (DeckGUI)
};
```

The complete DeckGUI.cpp

Note there might be slight variations between your version and this version.

```
// assumes your JuceHeader file is here
// check your MainComponent.h to see
// how the include is configured on your
// system
#include "../JuceLibraryCode/JuceHeader.h"
#include "DeckGUI.h"

//==============================================================================
DeckGUI::DeckGUI(DJAudioPlayer* _player) : player(_player)
{

    addAndMakeVisible(playButton);
    addAndMakeVisible(stopButton);
    addAndMakeVisible(loadButton);

    addAndMakeVisible(volSlider);
    addAndMakeVisible(speedSlider);
    addAndMakeVisible(posSlider);


    playButton.addListener(this);
    stopButton.addListener(this);
    loadButton.addListener(this);

    volSlider.addListener(this);
    speedSlider.addListener(this);
    posSlider.addListener(this);


    volSlider.setRange(0.0, 1.0);
    speedSlider.setRange(0.0, 100.0);
```

```
    posSlider.setRange(0.0, 1.0);

}

DeckGUI::~DeckGUI()
{
}

void DeckGUI::paint (Graphics& g)
{
    /* This demo code just fills the component's background and
    draws some placeholder text to get you started.

    You should replace everything in this method with your own
    drawing code..
    */

    g.fillAll (getLookAndFeel().findColour (ResizableWindow::backgroundColourId));   // clea

    g.setColour (Colours::grey);
    g.drawRect (getLocalBounds(), 1);   // draw an outline around the component

    g.setColour (Colours::white);
    g.setFont (14.0f);
    g.drawText ("DeckGUI", getLocalBounds(),
                Justification::centred, true);   // draw some placeholder text
}

void DeckGUI::resized()
{
    double rowH = getHeight() / 6;
    playButton.setBounds(0, 0, getWidth(), rowH);
    stopButton.setBounds(0, rowH, getWidth(), rowH);
    volSlider.setBounds(0, rowH * 2, getWidth(), rowH);
    speedSlider.setBounds(0, rowH * 3, getWidth(), rowH);
    posSlider.setBounds(0, rowH * 4, getWidth(), rowH);
    loadButton.setBounds(0, rowH * 5, getWidth(), rowH);

}

void DeckGUI::buttonClicked(Button* button)
{
    if (button == &playButton)
    {
        std::cout << "Play button was clicked " << std::endl;
        player->start();
```

```
    }
    if (button == &stopButton)
    {
        std::cout << "Stop button was clicked " << std::endl;
        player->stop();

    }
    if (button == &loadButton)
    {
        auto fileChooserFlags =
        FileBrowserComponent::canSelectFiles;
        fChooser.launchAsync(fileChooserFlags, [this](const FileChooser& chooser)
        {
            auto chosenFile = chooser.getResult();
            player->loadURL(URL{chosenFile});

        });
    }
}

void DeckGUI::sliderValueChanged (Slider *slider)
{
    if (slider == &volSlider)
    {
        player->setGain(slider->getValue());
    }

    if (slider == &speedSlider)
    {
        player->setSpeed(slider->getValue());
    }

    if (slider == &posSlider)
    {
        player->setPositionRelative(slider->getValue());
    }

}

bool DeckGUI::isInterestedInFileDrag (const StringArray &files)
{
std::cout << "DeckGUI::isInterestedInFileDrag" << std::endl;
return true;
}

void DeckGUI::filesDropped (const StringArray &files, int x, int y)
```

```
{
std::cout << "DeckGUI::filesDropped" << std::endl;
if (files.size() == 1)
{
    player->loadURL(URL{File{files[0]}});
}
}
```