MIDI plugins with JUCE

EECS 4462 - Digital Audio

September 17, 2020

YORK UNIVERSITÉ UNIVERSITY

Important class: AudioProcessor

- Base class for audio plugins
- Your plugin class must inherit from AudioProcessor

```
class Arpeggiator : public AudioProcessor
{   … }
```

- Must declare a global function called createPluginFilter() that returns an instance of your plugin

```
AudioProcessor* createPluginFilter()
{
    return new Arpeggiator();
}
```

C++ info

- Class constructors work similar to Java
  - Same name as the class
  - Can have overloaded versions
- C++ also has destructors
  - Run when an instance is destroyed
  - Same name as class with a ~ in front

```
~Arpeggiator() {}
```

C++ info

- Declaring an object in C++ is enough to create an object at run time

```
Arpeggiator arp;
```

- The above creates an Arpeggiator object
- Such an object gets destroyed automatically when out of scope
- To dynamically create objects, use pointers

```
Arpeggiator *arp;
arp = new Arpeggiator();
```

C++ info

- C++ has no garbage collection
- You must delete dynamically created objects manually

```
Arpeggiator *arp;
arp = new Arpeggiator();
delete arp;
```

- This will call the destructor before releasing the memory
- **`malloc`**, **`realloc`**, **`free`** etc. can also be used for dynamic memory allocation

Buffer processing

- In JUCE, processing takes place in buffers
- For audio plugins, this buffer contains a number of audio samples (more in this in a week or two)
- For MIDI plugins, the buffer contains the MIDI events that took place since the last buffer
- Time information is based on the sample rate, even in the case of MIDI
- The duration of a buffer is
  Sample Rate x Number of Samples in Buffer

Important function: prepareToPlay

- Called once before processing starts
- Can be used to initialize any variables in your plugin
- Also sets the Sample Rate

Important function: processBlock

- Called repeatedly
- All the processing (converting input to output) happens in its body
- Receives an AudioBuffer and a MidiBuffer
  - Only one of them will contain data based on the type of the plugin
- Timing information is obtained from the AudioBuffer even in the case of a MIDI plugin

C++ info

- Static functions in C++ are similar to static methods in Java, but the syntax is a bit different

```
MidiMessage::noteOff (1, lastNoteValue)
```