

# Object Oriented Programming


## Coursework for Endterm: Ocodecks

### Introduction

In this course, we have developed a basic DJ application called Ocodecks. For the end of term assessment, you are tasked with developing the application further by adding a custom deck control Component and a music library Component, then integrating them into a new GUI layout of your own design. The custom deck control Component should have custom graphics, implemented in the paint function and it should offer a means to control a deck in some interesting way of your own design. The music library component should allow the user to manage a library of music within the application. They should be able to search the music library and load music from it into the decks. It should also persist between application loads, so it will need to store its state in a data file.

### Requirements

We will assess your work based on the following requirements and criteria:

- R1: The application should contain all the basic functionality shown in class:
  - R1A: can load audio files into audio players
  - R1B: can play two or more tracks
  - R1C: can mix the tracks by varying each of their volumes
  - R1D: can speed up and slow down the tracks
- R2: Implementation of a custom deck control Component with custom graphics which allows the user to control deck playback in some way that is more advanced than stop/ start.
  - R2A: Component has custom graphics implemented in a paint function 
  - R2B: Component enables the user to control the playback of a deck somehow
- R3: Implementation of a music library component which allows the user to manage their music library
  - R3A: Component allows the user to add files to their library
  - R3B: Component parses and displays meta data such as filename and song length
  - R3C: Component allows the user to search for files
  - R3D: Component allows the user to load files from the library into a deck
  - R3E: The music library persists so that it is restored when the user exits then restarts the application
- R4: Implementation of a complete custom GUI
  - R4A: GUI layout is significantly different from the basic DeckGUI shown in class, with extra controls

- R4B: GUI layout includes the custom Component from R2
- R4C: GUI layout includes the music library component from R3

## Code style and technique

Your code should be written according to the following style and technique guidelines:

- C1: Code is organised into header (.h or .hpp) files and implementation files (.cpp). Header files contain class interface definitions, cpp files contain implementations of class function members.
- C2: Class interfaces in header files have comments for each public function describing purpose, inputs and outputs
- C3: Code is laid out clearly with consistent indenting
- C4: Code is organised into functions with clear inputs and outputs and a clear, limited purpose
- C5: Code is stateless wherever possible – functions make use of data passing in preference to global or class scope data.
- C6: Functions, classes and variables have meaningful names, with a consistent naming style
- C7: Functions do not change the state of class or global scope variables unless that is the explicit purpose of a function (e.g. a setter)

## Documentation

You should write a report and submit your source code. The submission should contain the following items and information:

- D1: Source code in standard ZIP format
- D2: Report in PDF format
- D3: Requirements: for each sub requirement (R1A → R4C) state how this was achieved or if it was not achieved. Explain where it can be found in the code. Use focused, short code extracts if they make your explanation clearer. Use annotated screenshots to describe your GUI components
- D4: Video tutorial: please upload a short video tutorial (5 minutes long) aimed at budding DJs who might want to use your software. Show how the user can load files into the players, how they can use the playback controls and any other features such as the music library and your custom deck control.

## Marking criteria

We will mark your work according to the set of criteria shown below, which consider the requirements, your programming technique and style and the documentation you have provided:

Category	Criterion	Not addressed	Attempted but did not meet requirements	Met requirements well but did not go beyond that	Met requirements well and went significantly beyond them.
Code style and	C1: Code is organised into				

technique	header (.h or .hpp) files and implementation files (.cpp). Header files contain class interface definitions, cpp files contain implementations of class function members.				
Code style and technique	C2: Class interfaces in header files have comments for each public function describing purpose, inputs and outputs				
Code style and technique	C3: Code is laid out clearly with consistent indenting				
Code style and technique	C4: Code is organised into functions with clear inputs and outputs and a clear, limited purpose				
Code style and technique	C5: Code is stateless wherever possible – functions make use of data passing in preference to global or class scope data.				
Code style and technique	C6: Functions, classes and variables have				

	meaningful names, with a consistent naming style				
Code style and technique	C7: Functions do not change the state of class or global scope variables unless that is the explicit purpose of a function (e.g. a setter)				
R1: Basic functionality	R1A: can load audio files into audio players				
	R1B: can play two or more tracks				
	R1C: can mix the tracks by varying each of their volumes				
	R1D: can speed up and slow down the tracks				
R2: Custom deck control Component	R2A: Component has custom graphics implemented in a paint function				
	R2B: Component enables the user to control the playback of a deck somehow				
R3: Music Library	R3A: Component allows the user				

	to add files to their library				
	R3B: Component parses and displays meta data such as filename and song length				
	R3C: Component allows the user to search for files				
	R3D: Component allows the user to load files from the library into a deck				
	R3E: The music library persists so that it is restored when the user exits then restarts the application				
R4: Custom GUI	R4A: GUI layout is significantly different from the basic DeckGUI shown in class, with extra controls				
	R4B: GUI layout includes the custom Component from R2				
	R4C: GUI layout includes the music library component from R3				

Document ation	D1: source code as zip				
	D2: report as PDF				
	D3: reporting on requirements				
	D4: Video tutorial				