KeyChord

- Virtual Piano Studio

CS5004 Final Project

Team Members: Jiaxin Jia, Xiaoyuan Lu

Why KeyChord? Addressing the Gaps

1

The Problem: Existing Solutions

Current virtual piano applications are often either too simplistic (mouse-only toys) or overly complex professional Digital Audio Workstations for the average user.

2

Our Goal: Accessibility & Efficiency

We aimed for a lightweight, keyboard-centric experience that provides instant gratification. Our vision was to make music creation accessible without the need for expensive hardware.

Foundation: Essential Tools

Virtual Keyboard

A 25-key range (C3-C5) supporting both keyboard and mouse input for versatile playing.

Recording Manager

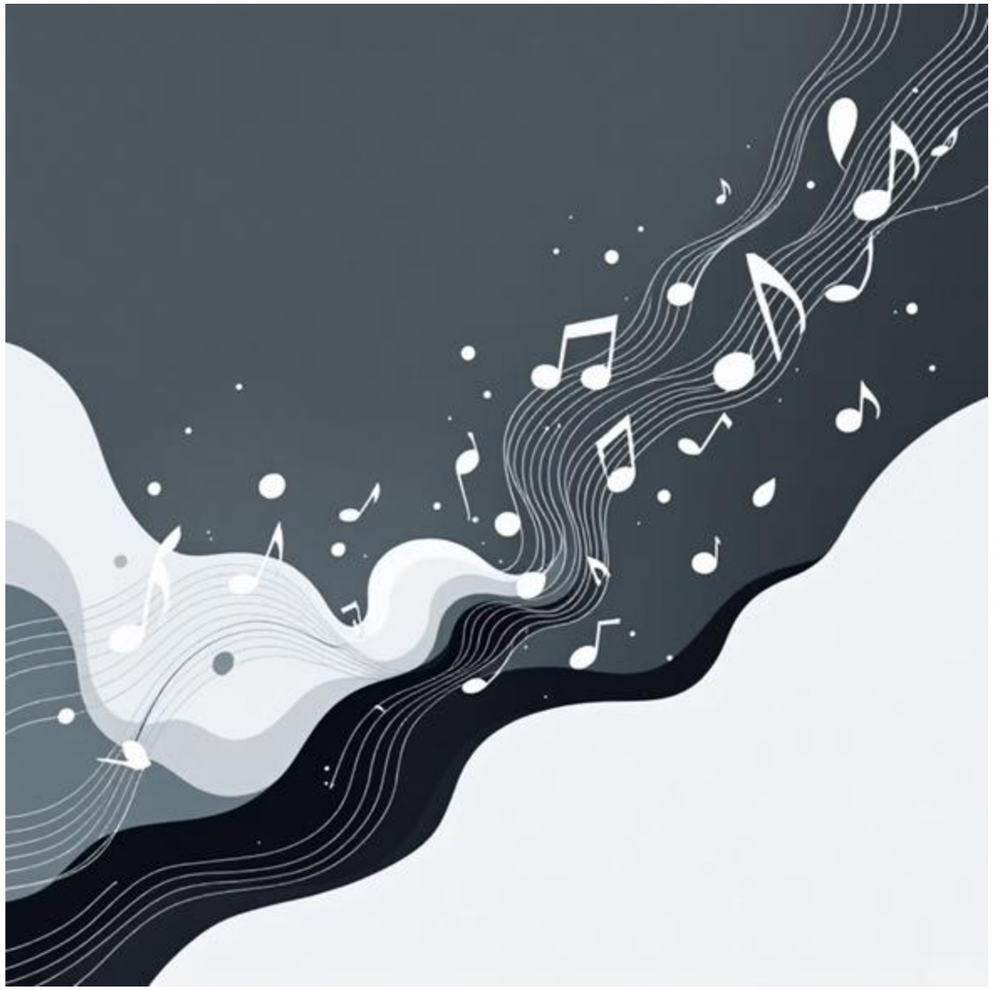
Seamlessly record, rename, delete, and playback your musical performances with ease.

MIDI Export

Export your recordings as standard `.mid` files, allowing integration with other music software.

Innovation: Smart Harmony Engine

- **Instant Harmony:** Transform a single key press into a rich, complex harmony.
- **Versatile Modes:** Choose from Single Note, Major, Minor, 7th, Sus2, and Sus4 chords.
- **Intelligent Algorithm:** Dynamically calculates intervals (e.g., Root + 4 + 7 semitones) for accurate chord voicings.
- **Professional Sound:** Achieve a sophisticated sound with zero music theory knowledge required, making advanced harmonies accessible to everyone.



KeyChord in Action: Live Demonstration

0

Octave Mapping Fix

Experience the full 25-key range in action, showcasing improved responsiveness.

02

Smart Chords

A dynamic demonstration of the 'Major 7th' mode, highlighting effortless harmony generation.

03

Recording Workflow

Observe the seamless process of recording a piece, followed by synchronized visual playback.

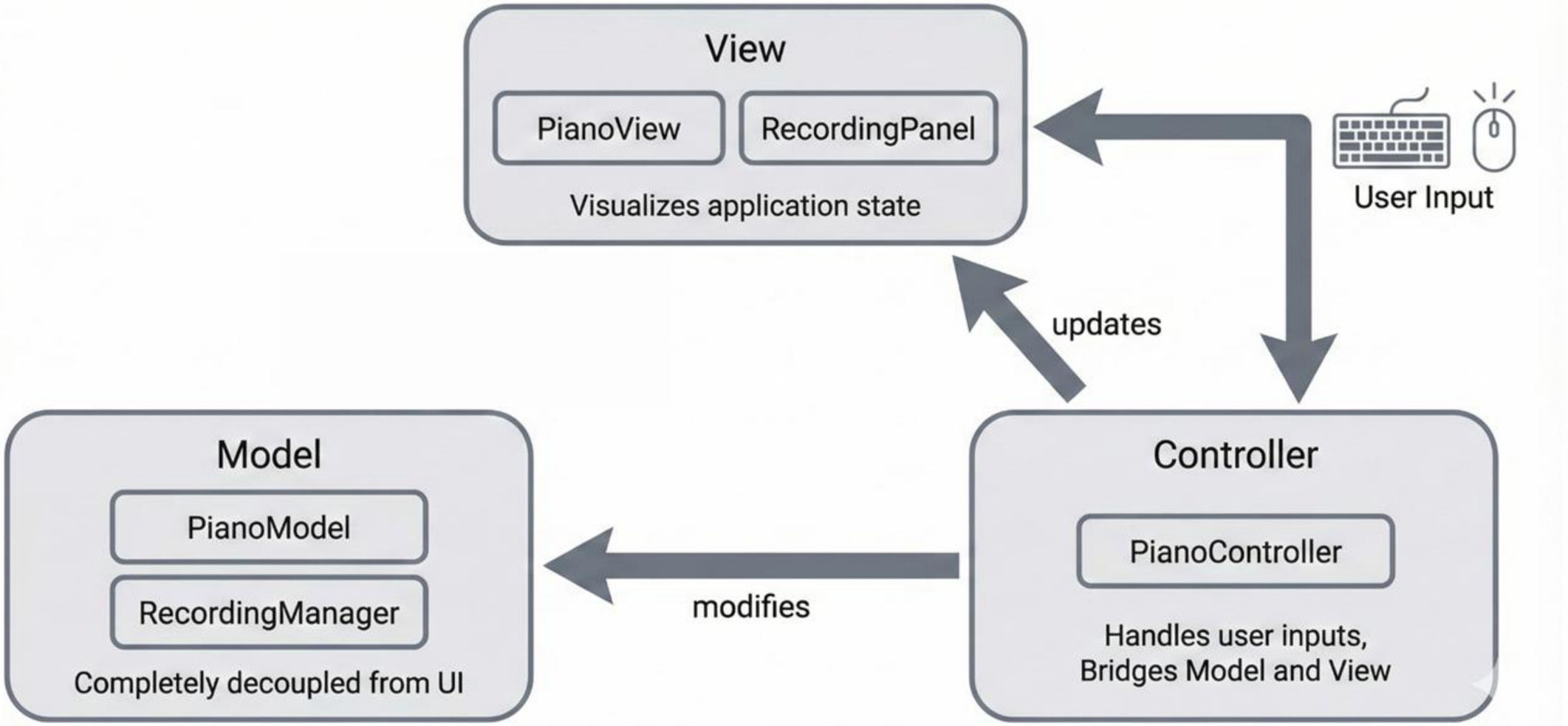
04

Track Management

Illustrating the intuitive renaming and organisation of recorded tracks within the application.

(Live Software Demonstration will follow)

Architecture Overview: Strict MVC Pattern



SOLID Principles Applied for Robust Design

Single Responsibility Principle (SRP)

The `Recorder` class exclusively manages timestamps, while the `Player` class is solely responsible for playback operations.

Open/Closed Principle (OCP)

`ChordManager` utilises an Enum Strategy; adding new chord types requires no modifications to existing logic.

Dependency Inversion

`PianoController` depends on abstractions (`PianoModel` interface), not concrete implementations.

Algorithmic Logic: Smart Chord Generation Precision

Strategy Pattern

Logic is decoupled from chord definitions

Open/Closed Principle:

Add new chords (e.g., Jazz, dim7) without changing core algorithm logic

Dynamic Calculation

Uses interval arrays (e.g., `[0, 4, 7]`) instead of hardcoded notes.

Safety

Automatic boundary checks (0-127 MIDI range)

Algorithmic Logic: Smart Chord Generation Precision

```java
public List<Integer> generateChord(int rootNote, ChordType chordType) {
    // Design Principle: Input Validation - validate MIDI note range
    if (rootNote < 0 || rootNote > 127) {
        throw new IllegalArgumentException("Root note must be between 0 and 127");
    }

    // Design Principle: Strategy Pattern - delegate to selected chord type
strategy
    int[] intervals = chordType.getIntervals();
    List<Integer> chordNotes = new ArrayList<>();

    for (int interval : intervals) {
        int note = rootNote + interval;
        // Clamp to valid MIDI range
        if (note >= 0 && note <= 127) {
            chordNotes.add(note);
        }
    }

    return chordNotes;
}
```

Data Structures: Optimizing for Performance

HashMap for Low Latency (The "Dictionary")

Usage: ***KeyMappings*** class maps keyboard characters to MIDI notes.

Rationale: We chose HashMap over iterating through lists.

Benefit: Provides O(1) lookup time. Crucial for a musical instrument where latency must be zero.

```
// Mapping from keyboard character to semitone offset from middle C
    private static final Map<Character, Integer> KEY_TO_OFFSET = new HashMap<>
();
    static {

        // White keys (Lower Octave Starts at C3, Offset -12)
        KEY_TO_OFFSET.put('q', -12); // C3
        KEY_TO_OFFSET.put('w', -10); // D3
        KEY_TO_OFFSET.put('e', -8);  // E3
```

ArrayList for Sequential Data (The "List")

Usage: ***Recording*** class stores sequences of NoteEvent.

Rationale: Music is inherently sequential and ordered.

Benefit: ArrayList preserves insertion order (time) and allows dynamic resizing as the user records longer songs.

```
public void addNoteEvent(NoteEvent event) {
        if (event == null) {
            throw new IllegalArgumentException("NoteEvent cannot be
null"); }
        synchronized (lock) {
            events.add(event);
        }
    }
```

Solving Complexity: Concurrency & Resources

Resource Contention

Problem: Limited MIDI System Resources.

Solution: Singleton Pattern (MidiSoundManager).

Benefit: Single global access point; prevents driver crashes.

Thread Safety

Problem: Concurrent access (UI Thread vs. Playback Thread).

Solution: Synchronization (synchronized locks).

Benefit: Prevents Race Conditions & "Stuck Notes".

```
public class MidiSoundManager {
    // Singleton Pattern
    private static MidiSoundManager instance;
    private static final Object instanceLock = new
Object();
    // Thread Safety - synchronized access
    public void playNote(int midiNote, int velocity) {
        synchronized (lock) {
            if (pianoChannel != null && initialized) {
                pianoChannel.noteOn(midiNote, velocity);
            }
        }
    }
}
```

Ensuring Robustness: Our Testing Strategy

1

The Challenge

UI events and MIDI dependencies are difficult to unit test directly

2

Our Strategy

Push logic into Model, keep View passive — MVC separation made testing significantly easier

3

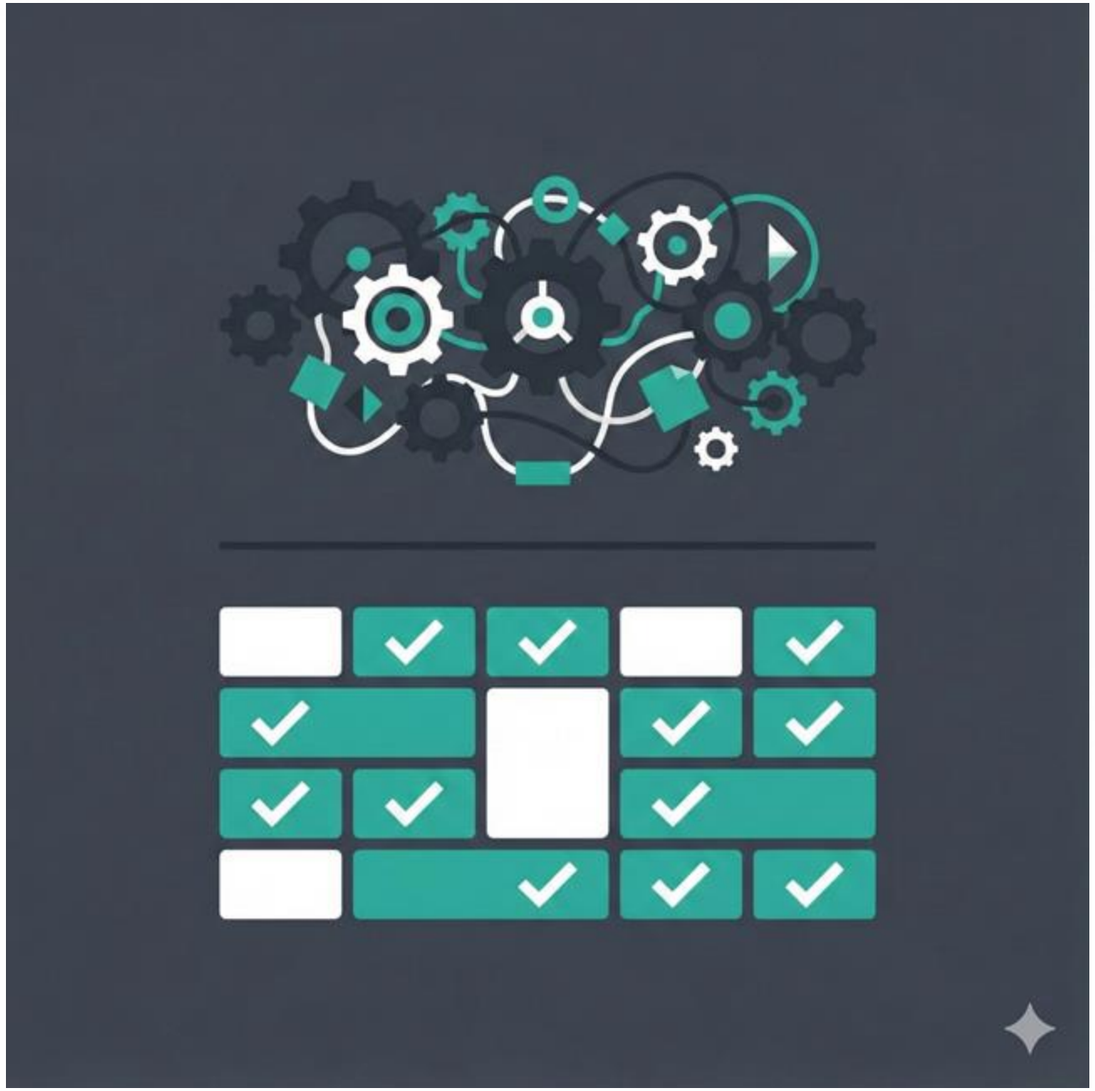
Core Test Areas

Smart Chord generation, RecordingManager state transitions, Key-to-note mappings

4

Tools & Results

JUnit 5 + Mockito to isolate dependencies; enabled confident refactoring without breaking workflows



Solving UI & UX Bugs: Polishing the User Experience

1

Layout Gaps

Fixed inconsistent whitespace issues by unifying all Panel heights to a consistent 200px, ensuring visual harmony.

2

Focus Stealing

Addressed the bug where clicking UI buttons would steal keyboard input from the piano, disrupting play.

3

The Fix: `setFocusable(false)`

Applied `setFocusable(false)` to buttons and `requestFocusInWindow()` to the piano, ensuring seamless interaction.

Result: A seamless and intuitive interaction between UI controls and piano playing, providing an uninterrupted creative flow for the user.

Lessons Learned

1

Technical Challenges

- Thread synchronization was more complex than expected
- Swing focus management required careful handling

2

Design Insights

- MVC separation made unit testing significantly easier
- Enum + Strategy pattern greatly improved extensibility

3

If We Started Over...

- Would implement data persistence earlier
- Would add a logging framework from the start

What's Next?

Future Enhancements

Persistence Capabilities

Implement saving recordings to local disk (JSON/XML) for data persistence across application restarts.

Multi-Instrument Support

Enable switching between various instrument sounds: Piano, Guitar, and Synth, for diverse musical expression.

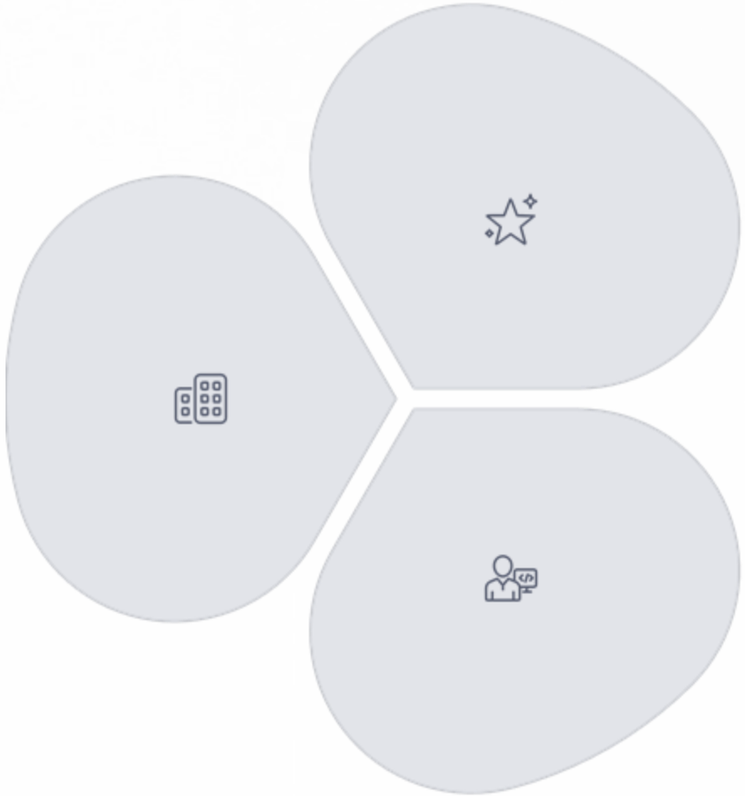
Advanced Visualiser

Integrate a waterfall-style note display to visually represent audio playback, enhancing user engagement.

Recap: Key Achievements

Architectural Excellence

Strict MVC implementation guarantees scalability and simplifies future testing and development.



Professional Value

Smart Chords and MIDI Export features offer genuine utility for musicians and developers alike.

Refined Process

Strategic refactoring, including deletion of legacy code, was pivotal to project success and clarity.

References

[1] Java MIDI API - docs.oracle.com/javase/8/docs/api/javax/sound/midi/

[2] Java Swing Tutorial - docs.oracle.com/javase/tutorial/uiswing/

[3] Java Concurrency - docs.oracle.com/javase/tutorial/essential/concurrency/

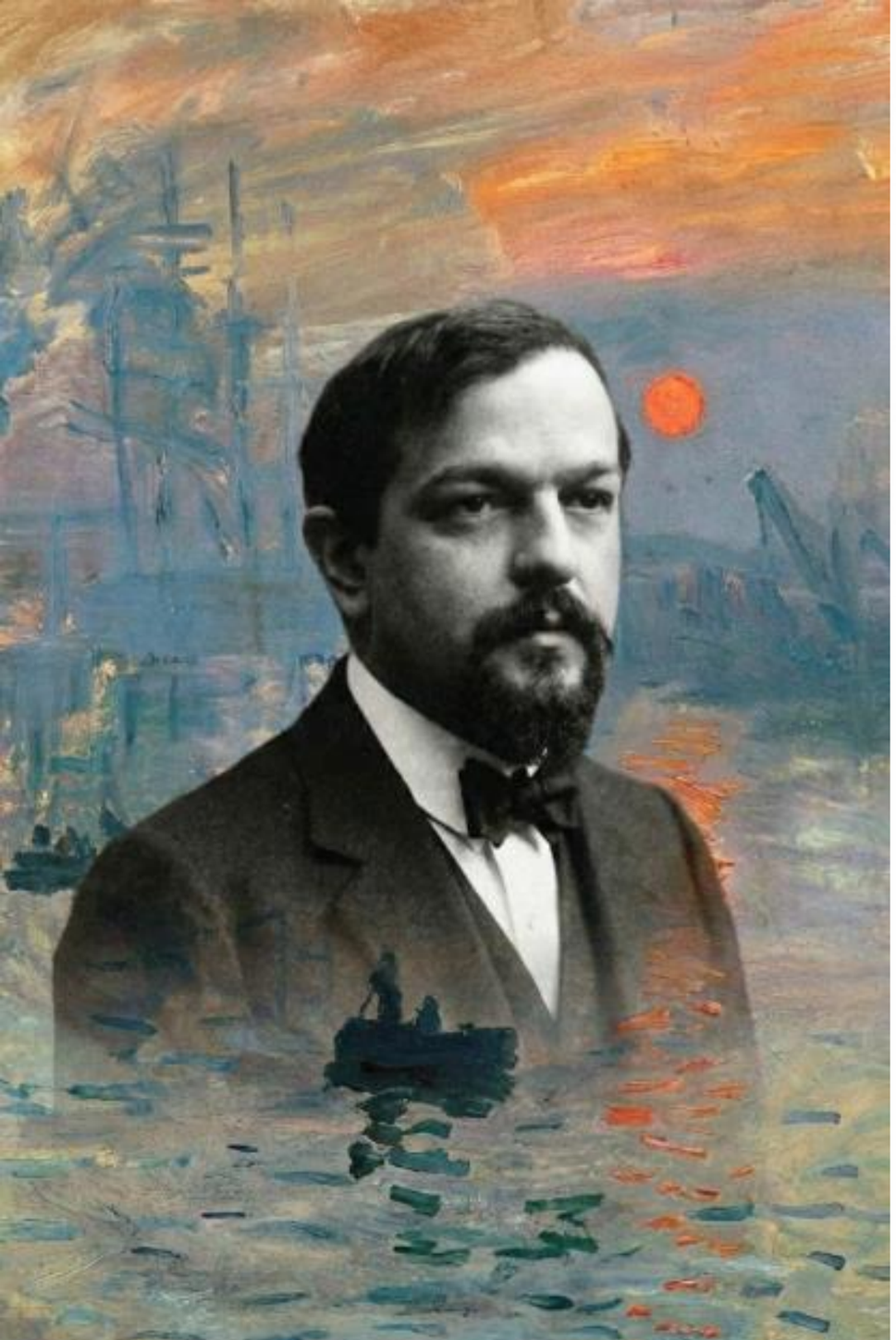
[4] JUnit 5 - junit.org/junit5/docs/current/user-guide/

[5] Mockito - site.mockito.org

[6] MIDI Specification - midi.org/specifications

[7] Music Theory - musictheory.net/lessons/40

[8] Design Patterns (Gamma et al., 1994)



Structure is what enables Beauty

> **"Music is the arithmetic of the soul."**
>
> **— Claude Debussy**

Structure & Flow: Just as OOD uses strict rules to create flexible software

Theory & Beauty: Music uses strict theory to create emotional art

Questions?