

Nova DAW Report & Manual

Inha University Java Application Programming (professor : wook-lee)

Nova daw (digital audio workstation)
program version : v1.0.0-beta | manual version : 1

컴퓨터공학과 12223818 조규연

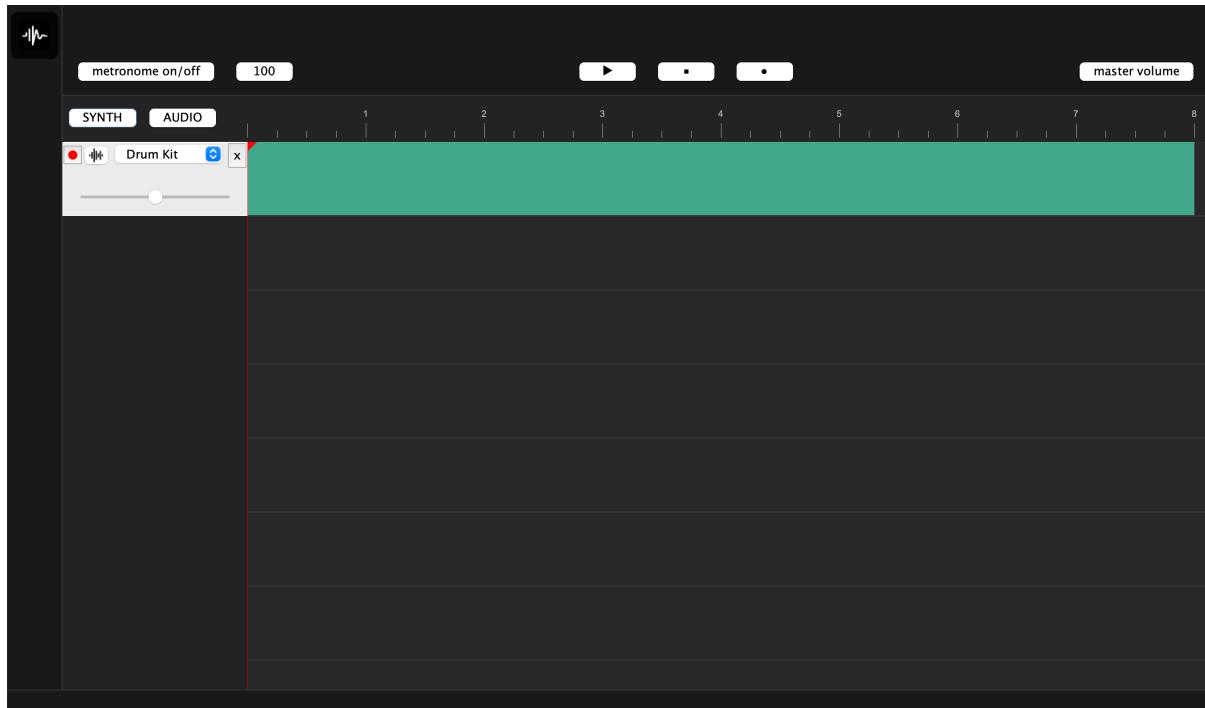


Audio craft

Table Of Contents

Project Abstract.....	3
Program Functional Requirements.....	4
Design Pattern.....	5
Implement Detail.....	6
Q & A (what I learned).....	10
Conclusion and Future Improvement.....	19
Idea Note.....	20
Appendix.....	21
Nova DAW Manual.....	22
Reference.....	23

Project Abstract



프로젝트 개요

Nova DAW는 누구나 쉽게 8마디 음악을 만들 수 있는 Digital Audio Workstation입니다. 이 버전에서는 최대 마디가 8마디이고, 최대 마디에 도달하면 다시 반복합니다. 기본 악기로는 신디사이저와 드럼 키트를 제공하며, 사용자는 노트북 키보드를 통해서 연주, 옥타브 조절이 가능합니다.

개발 목적

최근 음악&오디오 프로그래밍에 관심을 갖게 되었습니다. 그래서 이번 프로젝트를 통해 DAW의 동작 방식과 가상악기와 연결 법은 어떻게 되는지, 신디사이저가 소리를 내는 원리에 대해 학습하고, 저만의 프로그램을 만들고자 프로젝트 주제로 선정하였습니다.

개발 기간 : 2025.05.20 ~ 2025.06.21

사용 기술 : Java Swing + Jsyn 신디사이저 라이브러리 [링크](#)

- Jsyn 사용 이유 : 오디오 프로세싱 프로그램같은 경우, 대부분 c++로 개발을 하기 때문에 java 라이브러리가 많이 부족했고, 신디사이저 라이브러리가 Jsyn 말고 Beads 가 있었는데 실시간 오디오 녹음/처리에는 Jsyn이 더 적절하다고 판단하여 Jsyn을 선택하였습니다.

개발 도구 : eclipse

Program Functional Requirements

처음 프로그램을 설계했을 때 계획했던 필수 기능들입니다.

기능	개발 여부	기능	개발 여부
녹음	o	오디오 파일 첨부	x
재생	o	미디 에디터	o
신디사이저 악기	o	export	x
드럼 키트	o	템포조절	x
메트로놈	x		

[1] 녹음 & 재생 기능 (완)

녹음 버튼을 누르면 실시간으로 입력받는 음이 녹음이 되고, 재생 버튼을 누르면 녹음했던 음들이 연주됩니다.

[2] 신디사이저 악기 (완)

신디사이저의 기본 파형인 사각파, 삼각파, 톱니파, 사인파 중 하나를 선택하고, 증폭기의 Attack, Decay, Sustain, Release를 조절하여 사운드를 만듭니다.
마지막으로 frequency와 reverb를 통해 사운드 디자인을 완성합니다.

[3] 드럼 키트 (완)

각 키를 누르면 정해진 샘플이 연주됩니다. 드럼 키트에는 킥, 스네어, 하이앳, 클랩1, 클랩2가 포함됩니다.

[4] 메트로놈

녹음 버튼을 누르면 처음 4마디 카운트를 미리 준 후 녹음에 들어갑니다. 녹음 도중에는 계속해서 박자에 맞춰 메트로놈 소리가 나옵니다.

[5] 오디오 파일 첨부

외부의 오디오파일을 드래그앤파울 형식으로 가져올 수 있습니다. Audio 트랙에 올려놓으면 트랙이 오디오 파일을 분석하여 파형을 시각화해줍니다.

[6] 미디 에디터 (완)

트랙을 더블클릭하면 녹음한 노트(음)들을 수정할 수 있는 화면이 나옵니다. 마우스로 클릭하고 백스페이스를 누르면 노트가 삭제되고, 드래그하여 원하는 타이밍으로 노트를 이동시킵니다.

[7] export

최종 녹음본을 wav파일로 내보냅니다.

[8] 템포조절

템포조절 버튼을 클릭 후 상하로 움직여 템포를 조절합니다.

Design Pattern

```
cd/src; tree -L 1
```

```
.  
└── daw ..... 소스코드 저장소  
└── images ..... 이미지 저장소  
└── samples ..... 샘플 저장소
```

(빨간색 : Implements Detail에서 설명할 클래스, 이외의 클래스는 단순 panel 혹은 래퍼 역할)

```
cd/src/daw; tree -L 4
```

```
daw  
├── main  
│   ├── BottomBar.java ..... 프로그램 바텀 바  
│   ├── ControlBar.java ..... 컨트롤러(녹음, 재생 버튼 등)  
│   ├── Daw.java ..... DAW JFrame  
│   ├── Main.java ..... 부트스트랩  
│   ├── LayeredPaneTrackMain.java ..... 트랙 부분 레이어 팬  
│   └── TrackBar.java ..... 트랙 바(트랙 컨트롤러 컨테이너)  
└── component  
    ├── EditArea.java ..... 수정 영역  
    ├── PlayHead.java ..... 재생바(빨간 세로 줄)  
    ├── TrackRuler.java ..... 눈금  
    ├── navigation  
    │   └── Bar.java ..... 왼쪽 바 영역  
    └── track  
        ├── NewTrack.java ..... 새로운 트랙 버튼  
        ├── Note.java ..... 노트(음) 클래스  
        ├── TrackBody.java ..... 트랙 래인이 모여있는 트랙 바디  
        ├── TrackController.java ..... 트랙 컨트롤러(악기 선택, 볼륨 조절)  
        └── TrackLane.java ..... 각각의 트랙 래인  
└── synth  
    ├── BasicSynthesizer.java ..... 신디사이저 악기  
    ├── DrumKit.java ..... 드럼 키트 악기  
    ├── Inst.java ..... 악기 인터페이스  
    ├── KeyboardPlayer.java ..... 키보드 플레이어 클래스  
    └── SynthDialog.java ..... 신디사이저 가상악기 다이얼로그  
└── utils  
    └── Utils.java ..... 유저 인터페이스 설정을 위한 전역변수
```

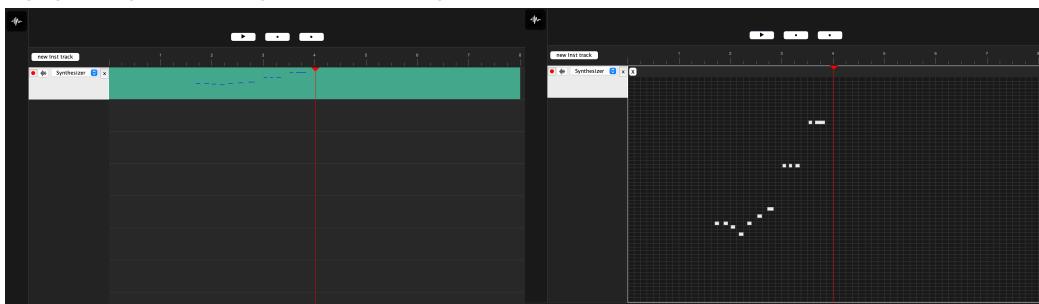
Implement Detail – ControlBar.java

```
public class ControlBar extends JPanel
```

구분	상세
역할	트랙 컨트롤러 – 재생버튼, 리셋버튼, 녹음버튼을 통해 녹음 과정을 컨트롤
생성자	<pre>ControlBar(PlayHead playhead) - 3개 버튼 추가 - 재생 버튼과 녹음버튼에는 StartPlayEventListener, 초기화 버튼에는 InitPlayEventListener 이벤트 각각 추가 - Playhead 인자를 받는 이유 : playhead 참조를 받아 setIsRecording(true)를 통해서 지금이 녹음중인지 그냥 재생중인지 를 정보를 저장</pre>
이벤트	<pre>StartPlayEventListener - 이벤트 클래스에 생성자를 추가하여 녹음시, 재생시 경우를 분리 - True이면 녹음중이라고 인식, actionPerformed 메서드에서 playhead.setIsRecording(true)로 녹음중임을 알림</pre> <div style="background-color: black; color: cyan; padding: 10px;"><pre>class StartPlayEventListener implements ActionListener { private boolean record; StartPlayEventListener(boolean record) { this.record = record; } @Override public void actionPerformed(ActionEvent e) { if(playhead.isPlaying() == false) { if(record == true) playhead.setIsRecording(true); else playhead.setIsRecording(false); playhead.startPlay(); } else playhead.stopPlay(); } }</pre><pre>InitPlayEventListener - Timer 멈추고 setPlayhead(0)로 Playhead 위치를 초기화한다.</pre></div>
모습	
피드백	원래 구현하려고 했던 메트로놈 기능과 템포 조절기능, 전체 볼륨 조절 기능이 추가되었으면 조금 더 컨트롤바 스러운 모습이었을 것 같다.

Implement Detail – DAW.java & Main.java

`public class DAW extends JFrame`

구분	상세
역할	모든 클래스들이 모이는 곳, JFrame
생성자	<p>DAW()</p> <p>전체 레이아웃은 다음과 같다.</p> <pre> BorderLayout ├─ WEST : Bar (navigation bar) ├─ CENTER : mainPanel (JPanel, BorderLayout) │ ├─ NORTH : ControlBar (상단 컨트롤) │ └─ CENTER : trackPanel (JPanel, BorderLayout) │ ├─ NORTH : newTrackAndRuler (JPanel, BorderLayout) │ ├─ WEST : NewTrack (트랙 추가 버튼 등) │ └─ CENTER : trackRulerScroll │ └─ trackRuler (시간 눈금자) │ ├─ CENTER : trackControlAndTrackLanes (JPanel, BorderLayout) │ ├─ WEST : trackBarScroll │ └─ trackBar (트랙 컨트롤러) │ └─ CENTER : layeredPane (LayeredPaneTrackMain, JLayeredPane) │ ├─ Z=3 : TrackBody (트랙 데이터 영역) │ ├─ Z=4 : EditArea (선택된 트랙 편집 UI) [viewEditor()에서 동적으로 추가] │ └─ Z=5 : PlayHead (재생 위치 표시 선) └─ SOUTH : BottomBar (하단 패널) </pre>
메서드	<p><code>public void viewEditor(TrackLane trackLane)</code></p> <ul style="list-style-type: none"> - 에디터 켜기 메서드 – 초록색 영역을 더블클릭 하면 에디터 켜짐 <p><code>public void closeEditor()</code> – 에디터 닫기 메서드</p> <p><code>public EditArea getEditArea()</code></p> <ul style="list-style-type: none"> - 에디터가 켜진 상태에서 녹음 시 TrackLane에서 입력을 받으면서 바로 바로 EditArea를 업데이트하기 위해 참조자를 받아오는 메서드
모습	에디터 꺼진 모습과 켜진모습 비교 
피드백	코드가 너무 복잡한데, 의존 관계가 얹혀서 코딩 후에는 쉽게 손을 댈 수가 없게 되었다. 앞으로는 개발 전 계획을 어느정도 디자인 패턴을 구상한다음, 처음부터 클래스를 잘 나눠서 개발해야겠다.

`public class DAW extends JFrame`

구분	상세
역할	프로그램부트스트랩, DAW 생성자를 호출하는 역할

Implement Detail – TrackBar.java

```
public class TrackBar extends JPanel
```

구분	상세
역할	여러 개의 트랙 컨트롤러가 담겨있는 컨테이너
생성자	<code>TrackBar(TrackBody trackBody)</code> <ul style="list-style-type: none">- <code>trackBody</code>를 생성자 입력으로 참조받아 트랙 바 안에 담겨있는 트랙 컨트롤러 중 클릭 녹음클릭(🔴) 이벤트가 발생한다면 <code>trackBody</code>에 focus할 수 있도록 한다.- 레이아웃을 BoxLayout – Y_AXIS로 함으로써 컨테이너 안에 여러 개의 트랙 컨트롤러가 Y축 방향으로 쌓일 수 있도록 한다.
메서드	<code>public Dimension getPreferredSize()</code> <ul style="list-style-type: none">- 초반에는 스크롤 기능이 들어가도록 코딩하였기 때문에 <code>scrolledPane</code>으로 감싸져있는데, 처음에 스크롤 안에 요소가 없을 때 화면에 나타나지 않는 문제가 있었음. 그래서 초기 크기를 <code>getPreferredSize()</code>를 오버라이딩 함으로써 크기를 결정함 [링크1] [링크2] <p>NewTrack.java 클래스에서 버튼 클릭시 호출하는 메서드이다. <code>public newTrack(TRACK_TYPE trackType)</code> – 새로운 트랙 생성</p> <p>트랙 컨트롤러에서 “포커스”, “악기변경” 등이 일어났을 때 트랙 컨트롤러에서 사용하는 메서드이다.</p> <ul style="list-style-type: none"><code>public void deleteTrack(TrackController track)</code> – 트랙 삭제<code>public void focus(TrackController tc)</code> – 트랙 focus<code>public void focus(int tc_idx)</code> – 녹음을 위해서 트랙 포커스<code>public void changeInst(int tc_idx, int index)</code> – 악기 변경하기<code>public Inst getInst(int tc_idx)</code> – 현재 선택된 악기 참조하기
모습	
피드백	원래는 악기가 많아지면 스크롤판을 통해서 위아래로 스크롤 할 수 있도록 하고 싶었는데 구조가 복잡해져서 스크롤 기능을 제거하고, 그에 맞게 악기는 8개만 추가할 수 있도록 하였다. 구조가 복잡해지는 이유: 트랙 바를 스크롤하면 Trackbody도 같이 스크롤 되어야 하지만, Trackbody를 좌우 스크롤 했을 때 트랙 바는 그대로 있어야 함

Implement Detail – EditArea.java

```
public class EditArea extends JPanel
```

구분	상세
역할	녹음했던 음들을 마우스로 수정할 수 있는 에디터 창
생성자	public EditArea(LayeredPaneTrackMain layeredPane, TrackLane trackLane)
메서드	<pre>public void initHeader : 에디터 창 닫기 버튼이 포함되어있다. public void initBody - 처음 에디터 창을 열었을 때 녹음 받은 데이터를 불러와 보여줌 public void refresh - 에디터창을 띄워놓고 녹음 받으면 실시간으로 에디터창에 노트가 찍히는 모습을 확인 가능하다. 이를 위해 키보드 Release 이벤트를 받을 때마다 에디터를 refresh하여 실시간 모습을 보여주었다.</pre>
이벤트	<pre>class EraseButtonListener extends KeyAdapter - 백스페이스 누르면 선택된 노트가 삭제된다. class MouseDragListener extends MouseAdapter - 노트(음)를 클릭하고 드래그하면 원하는 타이밍으로 노트를 움직일 수 있다.</pre>
모습	
그래픽	<p>배경의 격자무늬는 Graphics를 통해 그렸다. 각 블럭은 height=9 width=16를 가지고 있고, 에디터 영역 바로 위에 트랙 눈금과도 위치가 일치하도록 그렸다.</p> <pre>public void drawGrid(Graphics g, int blockHeight, int blockWidth) { g.setColor(Color.DARK_GRAY); for(int i=headerHeight; i<getHeight(); i+=blockHeight) { g.drawLine(0, i + blockHeight, getWidth(), i + blockHeight); } g.setColor(Color.DARK_GRAY); for(int i=0; i<16 * 2 * 4 * 8; i+=blockWidth) { g.drawLine(i + blockWidth, 0, i + blockWidth, getHeight()); } }</pre>
피드백	노트 삭제, 이동 뿐만 아니라 노트 길이 수정, 혹은 새로운 노트 생성 기 능도 구현하면 사용자 입장에서 더욱 편리할 것 같다.

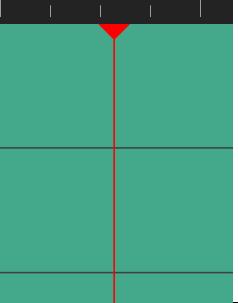
Implement Detail – EditArea.java

```
public class EditArea extends JPanel
```

구분	상세
역할	녹음했던 음들을 마우스로 수정할 수 있는 에디터 창
생성자	public EditArea(LayeredPaneTrackMain layeredPane, TrackLane trackLane)
메서드	<pre>public void initHeader : 에디터 창 닫기 버튼이 포함되어있다. public void initBody - 처음 에디터 창을 열었을 때 녹음 받은 데이터를 불러와 보여줌 public void refresh - 에디터창을 띄워놓고 녹음 받으면 실시간으로 에디터창에 노트가 찍히는 모습을 확인 가능하다. 이를 위해 키보드 Release 이벤트를 받을 때마다 에디터를 refresh하여 실시간 모습을 보여주었다.</pre>
이벤트	<pre>class EraseButtonListener extends KeyAdapter - 백스페이스 누르면 선택된 노트가 삭제된다. class MouseDragListener extends MouseAdapter - 노트(음)를 클릭하고 드래그하면 원하는 타이밍으로 노트를 움직일 수 있다.</pre>
모습	
그래픽	<p>배경의 격자무늬는 Graphics를 통해 그렸다. 각 블럭은 height=9 width=16를 가지고 있고, 에디터 영역 바로 위에 트랙 눈금과도 위치가 일치하도록 그렸다.</p> <pre>public void drawGrid(Graphics g, int blockHeight, int blockWidth) { g.setColor(Color.DARK_GRAY); for(int i=headerHeight; i<getHeight(); i+=blockHeight) { g.drawLine(0, i + blockHeight, getWidth(), i + blockHeight); } g.setColor(Color.DARK_GRAY); for(int i=0; i<16 * 2 * 4 * 8; i+=blockWidth) { g.drawLine(i + blockWidth, 0, i + blockWidth, getHeight()); } }</pre>
피드백	노트 삭제, 이동 뿐만 아니라 노트 길이 수정, 혹은 새로운 노트 생성 기 능도 구현하면 사용자 입장에서 더욱 편리할 것 같다.

Implement Detail – PlayHead & TrackRuler

public class PlayHead extends JPanel

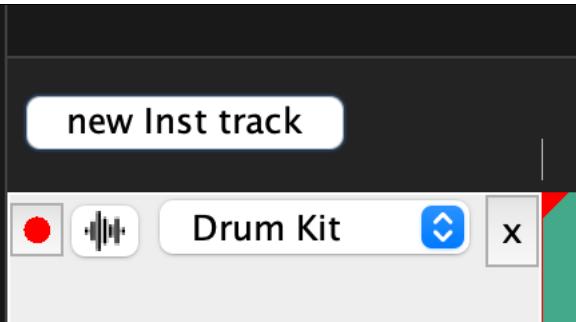
구분	상세
역할	재생바 : 현재 녹음이 되고있는 위치를 빨간 막대로 가리켜줌
생성자	public PlayHead() – 초기 위치 세팅, 타이머 초기화 - 10ms마다 현재 녹음된 소리를 연주한다.
메서드	public int getPosition() : 위치 반환 public boolean getIsPlaying() : 재생중인지 여부 반환 public void initPlay() : 처음 위치로 세팅 public void startPlay() : 재생하기 public void stopPlay() : 멈추기 public void setPlayhead(int position) : 위치 변경 후 다시 그리기 public void updatePlayhead() : 매 초마다 작동하는 함수, 위치 += 1 public void setIsRecording(boolean record) : 녹음 여부 세팅 public boolean isRecording() : 현재 녹음 중인지 반환한다.
모습	
그래픽	구조상 트랙바디, 에디터영역, 플레이헤드 이 세 요소가 레이어팬으로 겹치도록 구성함으로써 트랙 바디가 가장 아래, 에디터 영역이 그 위, 플레이 헤드가 가장 위쪽에서 보이도록 하였다. 또한 매 초마다 repaint를 해야하기 때문에 컴포넌트보다 가벼운 Graphics를 사용하여 빨간 선을 그렸다. <pre>protected void paintComponent(Graphics g) { super.paintComponent(g); g.setColor(Color.RED); g.drawLine(position, 0, position, 1000); int[] a = {position-10, position+10, position}; int[] b = {0, 0, 10}; g.fillPolygon(a, b, 3); }</pre>

public class TrackRuler extends JPanel

구분	상세
역할	재생바 : 현재 녹음이 되고있는 위치를 빨간 막대로 가리켜줌
모습	

Implement Detail – NewTrack.java

```
public class NewTrack extends JPanel
```

구분	상세
역할	새로운 트랙을 생성한다.
생성자	public NewTrack(ToolBar trackBar, TrackBody trackBody)
다이어로그	class MaxDialog extends JDialog - 최대 7개 트랙 생성 가능, 7개가 넘어갈 시 MaxDialog가 띄워짐
이벤트	class new TrackButtonEvent implements ActionListener - 버튼 클릭 시 트랙바에 트랙 컨트롤러를 추가하고, 트랙 바디에 는 트랙 래인(TrackLane – 초록색 영역)을 추가한다.
모습	

Implement Detail – Note.java

```
public class Note
```

구분	상세
역할	음 하나에 대한 정보를 담고있는 클래스
생성자	public Note(int key, int startTime, int endTime) - 무슨 음인지(key), 음의 시작 시간, 끝나는 시간
메서드	key, endTime, startTime 필드 각각에 대한 getter, setter 메서드

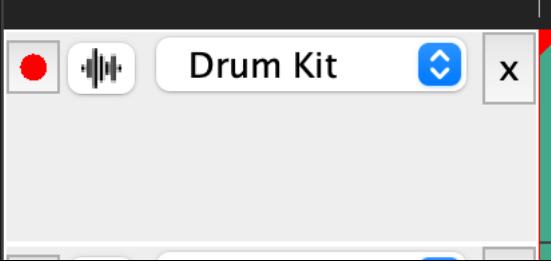
Implement Detail – TrackBody.java

```
public class TrackBody extends JPanel
```

구분	상세
역할	여러 트랙 래인(초록 영역)들을 감싸는 컨테이너
생성자	<pre>public TrackBody(PlayHead playhead, Daw daw)</pre> <ul style="list-style-type: none">- 각 트랙 래인을 더블 클릭 시 DAW 클래스의 viewEditor 메서드를 실행하기 위해 인자로 받아온다.- Playhead를 인자로 받아 현재 재생중인 위치(position)에 있는 노트들을 연주한다.
메서드	<pre>playCurrentNote() : position과 start_time이 겹치는 노트를 연주</pre> <pre>stopCurrentNote() : position과 end_time이 겹치는 노트 연주를 중단</pre>
모습	
피드백	<p>이 부분이 코드 짜기가 어려웠다. 현재 재생되는 위치를 얻기 위해 playhead 인스턴스를 참조받고, position 정보와 또 각각의 트랙 래인에 있는 노트와 비교를 하여 연주하고, 중지해야하기 때문이다.</p> <p>코드가 길어져서 복잡해보이는 감이 있는 것 같다.</p> <p>다음부터는 생성자 함수 안에서도 자식에게 인자를 넘겨주는 함수, 자신의 역할을 수행하는 함수 이렇게 로직을 나눠서 따로 함수로 구현한 뒤, 생성자 안에는 딱 메서드 몇 개 밖에 없도록 하여 유지보수가 편하도록 개발해야겠다.</p>

Implement Detail – TrackController.java

```
public class TrackBody extends JPanel
```

구분	상세
역할	트랙바에 안에 들어가는 요소, 해당 트랙의 악기, 녹음, 악기 세부 설정 버튼, 삭제 기능을 가지고 있다.
생성자	public TrackController(TRACK_TYPE trackType, - 나중에 확장을 위해 트랙의 타입을 오디오타입, 악기타입으로 나눌 수 있도록 하였다. (현재 버전은 악기타입만 가능) TrackBar trackBar, int tc_idx //생성 시 부여받는 번호)
모습	
버튼	녹음버튼 - 버튼을 누르면 해당 악기로 녹음이 가능하다. 버튼 클릭 시 해당 트랙의 트랙에 포커싱이된다. trackBar.focus(tc_idx); 웨이브 버튼 - 악기 세부 설정을 위한 버튼이다. 드럼 키트는 세부 이미 샘플이 정해져있기 때문에 세부 설정이 불가능하고, 신디사이저만 가능하다. 신디사이저 악기 선택 후, 웨이브 버튼 클릭 시 신디사이저 세부설정을 위한 다이어로그가 띄워진다. 악기 선택 콤보박스 - Drum Kit과 Synthesizer 중 선택할 수 있다. 선택 시 trachBar.changeInst(tc_idx, index); 가 실행된다. 닫기 버튼 : 해당 트랙을 삭제한다.

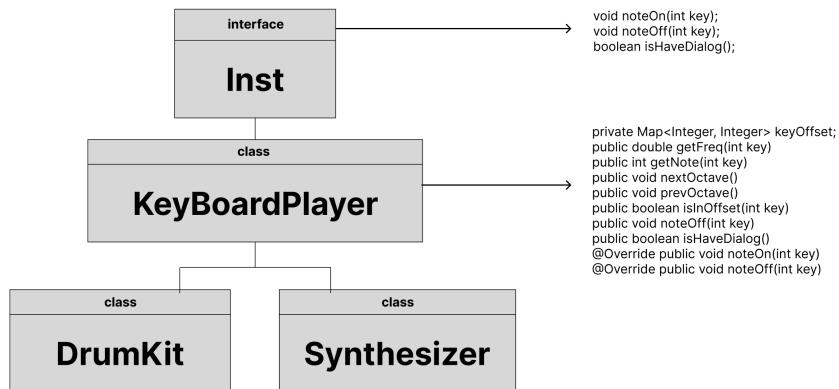
Implement Detail – TrackLane.java

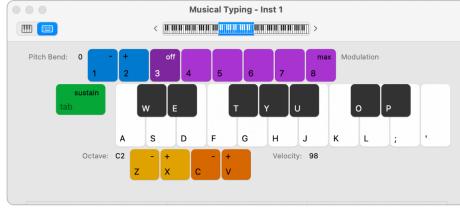
public class TrackLane extends JPanel

구분	상세
역할	연주된 노트들의 미리보기를 띄워주는 화면, 더블클릭 시 디테일한 조정을 할 수 있는 에디터창이 띄워진다. 또한 연주된 노트 벡터 배열이 저장되는 곳이기도 하다.
생성자	TrackLane(TRACK_TYPE trackType, PlayHead playhead, Daw daw)
멤버	<pre>private KeyboardPlayer inst : 현재 선택한 악기 private Map<Integer, Integer> keyStartTime - 현재 누르고 있는 노트의 시작 시간을 저장한다. - 키가 release되면 삭제된다. private Vector<Note> playData - 녹음된 음들을 저장하는 동적 배열(벡터)</pre>
구현 방식	<p>핵심이 되는 이벤트 리스터 클래스</p> <pre>class playKeyListener extends KeyAdapter { private Set<Integer> pressedKeys = new HashSet<>(); public void keyPressed(KeyEvent e) { int keyCode = e.getKeyCode(); if(pressedKeys.contains(keyCode)) return; if(keyCode == KeyEvent.VK_Z) inst.prevOctave(); else if(keyCode == KeyEvent.VK_X) inst.nextOctave(); inst.noteOn(inst.getNote(keyCode)); pressedKeys.add(keyCode); int start_time = playhead.getPosition(); System.out.println("Key pressed " + keyCode + " at position: " + start_time); keyStartTime.put(keyCode, start_time); } public void keyReleased(KeyEvent e) { int keyCode = e.getKeyCode(); int end_time = playhead.getPosition(); int start_time = keyStartTime.get(keyCode); pressedKeys.remove(keyCode); keyStartTime.remove(keyCode); Note note; playData.add(note = new Note(inst.getNote(keyCode), start_time, end_time)); if(playhead.isRecording() == true) playData.add(note = new Note(inst.getNote(keyCode), start_time, end_time)); inst.noteOff(inst.getNote(keyCode)); System.out.println("Key released " + inst.getNote(keyCode) + " at position: " + end_time); daw.getEditArea().refresh(); revalidate(); repaint(); } }</pre> <p>중복 입력을 막기 위해서 현재 누르고 있는 키를 저장하는 해시셋을 생성함</p> <p>키보드를 pressed 되었을 때</p> <ul style="list-style-type: none"> - 누른 키가 Z 혹은 X일 경우 옥타브 조절, 혹은 음 연주 - keyStartTime 해시맵에 put <p>키보드가 released 되었을 때</p> <ul style="list-style-type: none"> - keyStartTime에 저장되어있던 startTime과 키가 떨어진 시점인 endTime을 Note 생성자의 인자로 전달, playData 배열에 삽입 - 트랙 래인에 실시간으로 그려지기 위해 revalidate, repaint 실행
모습	

Implement Detail – 신디사이저 파트 (1)

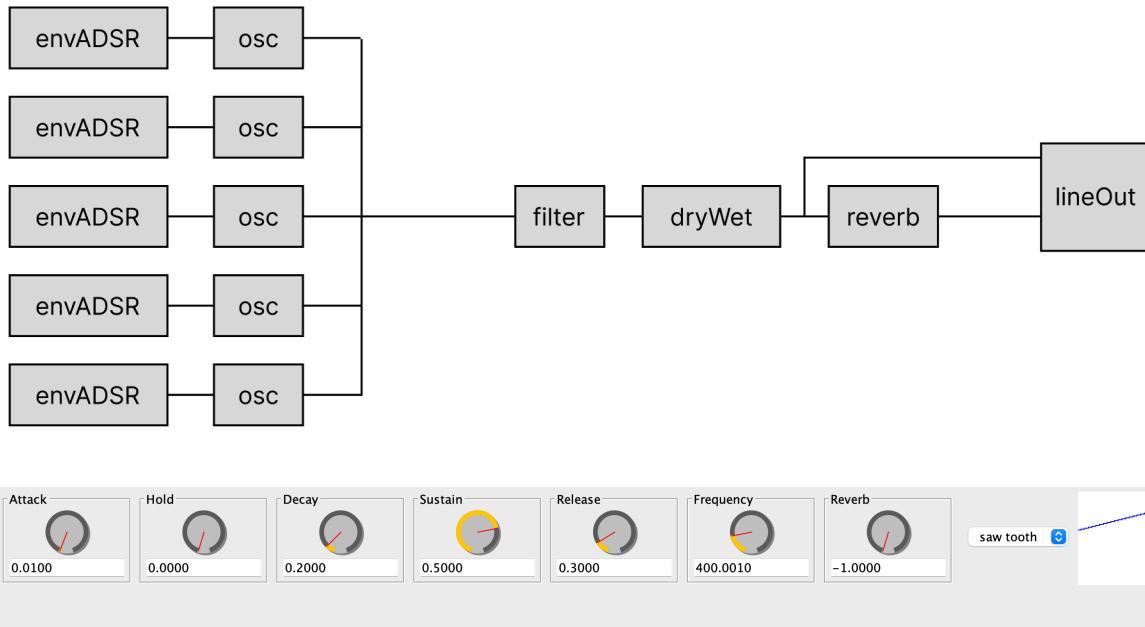
1. 클래스 계층도



구분	상세
계층 구조 목적	<p>최상위 인터페이스 Inst 까지 정해둔 이유</p> <ul style="list-style-type: none"> - 키보드 뿐만 아니라 디제잉 패드로의 확장성 고려  <p>DrumKit와 Synthesizer를 KeyBoardPlayer로 묶은 이유</p> <ul style="list-style-type: none"> - 둘 다 키보드를 입력으로 받아 연주되는 악기이기 때문
키보드 입력 방식	<p>사용자의 편리성을 위해 DAW 표준으로 사용되고 있는 키보드 배치를 사용하였다.</p>  <pre> void setKeyOffset() { keyOffset = new HashMap<Integer, Integer>(); keyOffset.put(KeyEvent.VK_A, 0); keyOffset.put(KeyEvent.VK_W, 1); keyOffset.put(KeyEvent.VK_S, 2); keyOffset.put(KeyEvent.VK_E, 3); keyOffset.put(KeyEvent.VK_D, 4); keyOffset.put(KeyEvent.VK_F, 5); keyOffset.put(KeyEvent.VK_T, 6); keyOffset.put(KeyEvent.VK_G, 7); keyOffset.put(KeyEvent.VK_Y, 8); keyOffset.put(KeyEvent.VK_H, 9); keyOffset.put(KeyEvent.VK_U, 10); keyOffset.put(KeyEvent.VK_J, 11); keyOffset.put(KeyEvent.VK_K, 12); keyOffset.put(KeyEvent.VK_O, 13); keyOffset.put(KeyEvent.VK_L, 14); keyOffset.put(KeyEvent.VK_P, 15); keyOffset.put(KeyEvent.VK_SEMICOLON, 16); keyOffset.put(KeyEvent.VK_QUOTE, 17); } </pre> <p>애플의 로직 Pro에서 지원하는 키노트이고, 오른쪽 코드는 키보드 입력을 노트로 변환하는 해시맵이다.</p>
주파수 계산	<p>KeyBoardPlayer의 경우 60을 중간 도로 기준 삼아, +1 -1 마다 음이 반음씩 증감된다. 신디사이저의 경우 이를 주파수 값으로 바꿔주어야 하기 때문에 주파수 계산 함수를 구현하였다.</p> $f(n) = 440 * 2^{(n-69)/12}$ <pre> public double getFreq(int key) { double freq = 440.0 * Math.pow(2.0, (key - 69) / 12.0); System.out.println(key); return freq; } </pre>

Implement Detail – 신디사이저 파트 (2)

2. 신디사이저 구조



Q & A (what I learned)

Q. 개발하면서 가장 어려웠던 점? # 디자인 패턴

개발의 첫 시작단계였던 것 같습니다. Java swing을 이용한 daw 레퍼런스 코드를 찾지 못해서 구조를 짜는 단계가 가장 막막했습니다. 당시 Layout에 대한 이해도 서툴었기 때문에 처음 UI를 구현하는데 까지 1주일 넘게 사용했습니다. 그렇기에 BorderLayout에 익숙해지고 그 안에서 일어나는 크기 설정에 대해 이해를 한 후 제대로된 개발을 시작할 수 있었습니다.

하지만 이렇게 무작적 코드부터 짜는 코딩으로 인해 개발 후반부에 가서 로직이 꼬이기 시작했습니다. 어떠한 [클래스에서 다른 클래스의 객체를 참조해야 할 일이 생길 때 생성자를 통해 객체를 통째로 넘겨주는 방식에](#) 로직이 너무 복잡해졌습니다. 그 이유는 처음에 하나의 클래스나 하나의 함수 안에 모든 내용을 다 집어 넣었기 때문이었습니다.

이를 해결하기 위해 [코드가 길어지는 부분이 있으면 따로 클래스를 분리하였고,](#) 개발 전 어떻게 코딩할 것인지 [디자인 패턴을 잘 계획하는 것이 중요함을](#) 느꼈습니다.

Q. 프로젝트에서 배운 점? # docs 읽는 법 # 머릿속 생각을 바로 코딩하기

신디사이저 구현을 위하여 Jsyn 라이브러리를 공부해야했습니다. 이전까지는 새로운 라이브러리나 기술을 배울 때 한글로된 책이나 영상과 같은 가이드가 무조건 필요했습니다. 이해하기도 쉬웠고, 예제 코드가 있었기에 제가 인터넷을 뒤져볼 필요도 없었기 때문입니다. 하지만 Jsyn 라이브러리는 책은 물론 유튜브 영상 강의도 없었습니다. 단지 문서와 사이트에 있는 영어로된 예제 설명이 전부였기에 학습에 어려움이 있었습니다.

저의 공부 방법은 [github 상에 올라와있는 example 코드를 일단 무작정 필사하는](#) 것이었습니다. 이후 [document를 통해서 상속 관계를 파악하였습니다.](#) 부모 클래스에는 어떤 메소드가 있어야겠구나 예측하고 그 예측을 확인해보는 방식이었습니다. 또한 왜 안 될까 하는 경우에도 document 상속관계를 파악하면 그 이유를 알게 되는 경우도 많았습니다.

이를 통해서 처음 보는 [라이브러리나 기술 문서를 접할 때 두려움이 사라졌고](#), 오히려 상속 계층도를 보면 마음이 편해졌습니다. 이는 [java 뿐만 아니라 다른 개발 프로젝트에서도 많은 도움이 되었습니다.](#)

교수님께서 docs 읽는 방법을 아는 것에 대해 강조했던 이유를 알게 되었고, 이는 저에게 큰 밑거름이 되었습니다. 감사합니다.

Q. 앞으로 하고 싶은 것? # VST 개발(c++) # Audio AI

DAW와 가상악기는 보통 성능상의 이유로 C++로 개발합니다. 방학 때는 C++로 저만의 가상악기를 만들어보고 싶습니다. 오디오 처리 분야는 개발 뿐만 아니라 신호처리, 신디사이저 지식이 필요하기 때문에 그 분야도 골고루 공부하고 싶습니다.

추가로 오디오 처리 분야 AI에 대해 공부하고 이를 가상악기에 접목시켜 음악 소스를 생성하는 AI를 개발해보고 싶습니다. 최근 이 분야 유명 브랜드인 Waves사에서 이러한 제품을 개발하여 더욱 관심을 갖게 되었습니다.

Conclusion and Future Improvement



Audio Craft

본 프로젝트는 기존 DAW과 visual studio code의 디자인에서 영감을 얻어 나만의 DAW를 만들어보겠다는 목표로 진행되었습니다. 핵심 기능인 녹음, 재생, 에디터, 신디사이저, 드럼 키트 등의 기능은 잘 구현 되었고 프로그램을 이용해 저만의 음악을 만들 수 있었습니다.

다만 신디사이저 라이브러리 학습 과정에서 시간이 많이 소모되어 나머지 계획했던 기능을 구현하지 못한 점에서 아쉬움이 남습니다. 그래도 라이브러리를 공부하는 과정에서 개발에 필요한 중요한 역량을 많이 쌓았습니다.

구현 요구사항 목록 중에서 아직 개발 하지 못한 부분을 완성하고, 이후에 추가로 AI 기능을 탑재하여 다른 DAW와의 차별점을 갖춘 어플리케이션을 만들고 싶습니다.

Idea Note

화면 설계도	<p>Border</p> <ul style="list-style-type: none"> • left • center (border) <ul style="list-style-type: none"> • top • left (border) <ul style="list-style-type: none"> • NORTH : 새 트랙 생성 버튼 • SOUTH : 별간영역 • right <ul style="list-style-type: none"> • NORTH : 트랙 높급 • SOUTH : 트랙 영역 • bottom - (숨기기버튼 클릭시 사라짐) • bottom - 설정 영역
초기 디자인	
아이디어 노트	<p>Border</p> <ul style="list-style-type: none"> • left • center (border) <ul style="list-style-type: none"> • top • left (border) <ul style="list-style-type: none"> • NORTH : 새 트랙 생성 버튼 • SOUTH : 별간영역 • right <ul style="list-style-type: none"> • NORTH : 트랙 높급 • SOUTH : 트랙 영역 • bottom - (숨기기버튼 클릭시 사라짐) • bottom - 설정 영역

Appendix

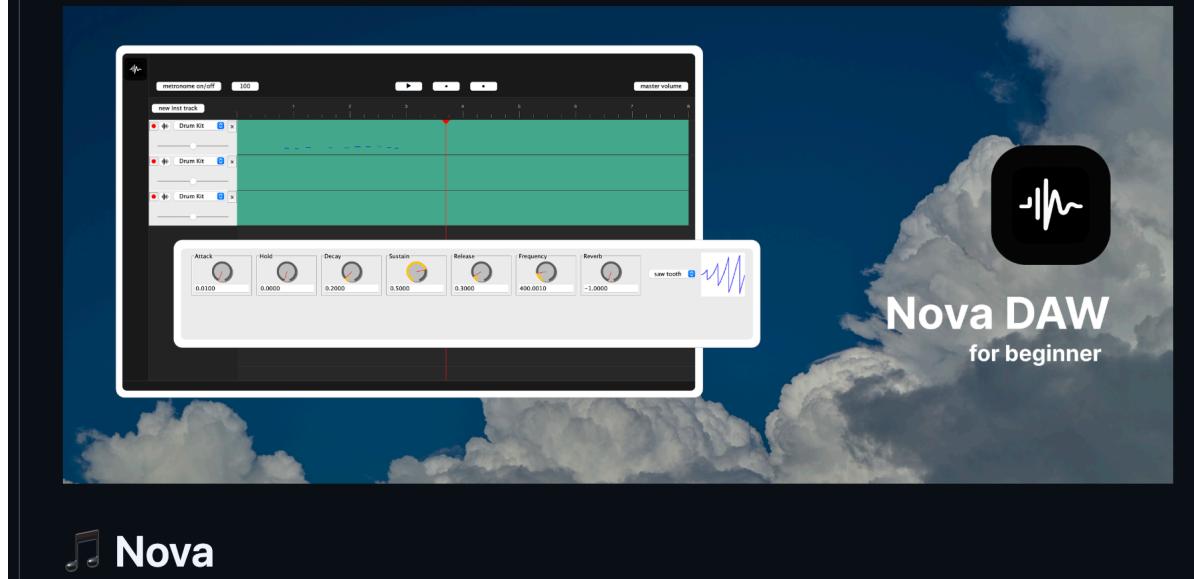
신디사이저 개발 공부 과정 [블로그 링크]

[JSyn 자바] 튜토리얼 한국어 설명(1) ★★★★★	0	방금 전
[JSyn 자바] JSyn 오실레이터를 파봅시다.	8	2025. 6. 6.
[JSyn 자바] 신디사이저(enveloper ADSR 기능)	2	2025. 6. 6.
[JSyn 자바] 샘플러와 멀티 보이스 신스 가능 가지고 드럼 키트 만들기 (1)	2	2025. 6. 2.

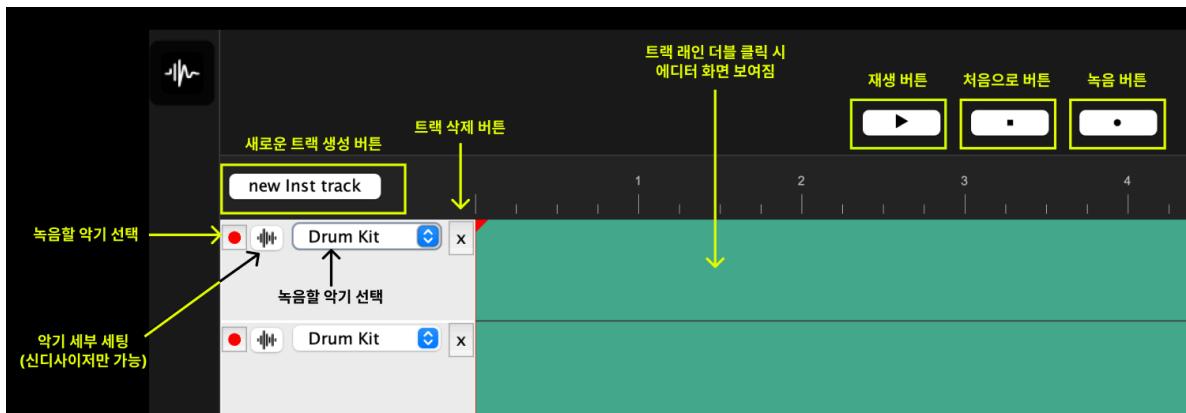
Github 레포지토리 [깃허브 링크]

 rbdus0715 Update README.md	07fc3d3 · 1 minute ago	 5 Commits
📁 .settings first commit		yesterday
📁 bin feat - synthesizer dialog		13 minutes ago
📁 libs first commit		yesterday
📁 src feat - synthesizer dialog		13 minutes ago
📄 .classpath first commit		yesterday
📄 .project first commit		yesterday
📄 README.md Update README.md		1 minute ago

README



Nova DAW Manual



Reference

디자인 참고

- [1] Logic pro daw <https://www.apple.com/kr/logic-pro/>
- [2] Bandlab daw <https://www.bandlab.com/>
- [3] EchoInMirror <https://github.com/eimsound/EchoInMirror>

공부한 곳

- [1] 신디사이저 원리와 gui 구현에 대한 아이디어 [\[유튜브 링크\]](#)
- [2] Jsyn example code 참고 <https://github.com/philburk/jsyn>
- [3] Jsyn document <https://www.softsynth.com/jsyn/docs/javadocs/>
- [4] Jsyn userguide <https://www.softsynth.com/jsyn/docs/usersguide.php>
- [5] 미디 노트 - 주파수 변환 함수
<https://www.music.mcgill.ca/~gary/307/week1/node28.html>

나중에 읽을거리

- [1] 미디 노트 - 주파수 변환 함수의 업계 표준에 대한 토론
<https://forums.steinberg.net/t/midi-note-to-hz-relation-can-it-change-is-it-an-industry-standard/131089>
- [2] 한양대학교 음악 프로그래밍 수업 강의노트 <https://doggzone.github.io/cse2020/>

참고

- [1] 신디사이저 엔벨로프 ADSR

