실력만은 20학번

문제해결기법 최종보고서

03조

팀 장		
	최인호	16011039
팀 원		
	안건우	10611029
	박기춘	16011035
	김성수	16011021

목차

프로젝트 소개 1

팀원 별 리펙토링 내용

- 최인호 3
- 안건우 8
- 박기춘 15
- 김성수 21

개인별 리펙토링 세부 내용

1. 담당 파트
2. 일정
3. 리펙토링 내용
4. 개인소감

1. 프로젝트 소개

프로젝트 의의

각 사이트의 TOP 100 차트 및 앨범 별 댓글을 보여주는 프로그램

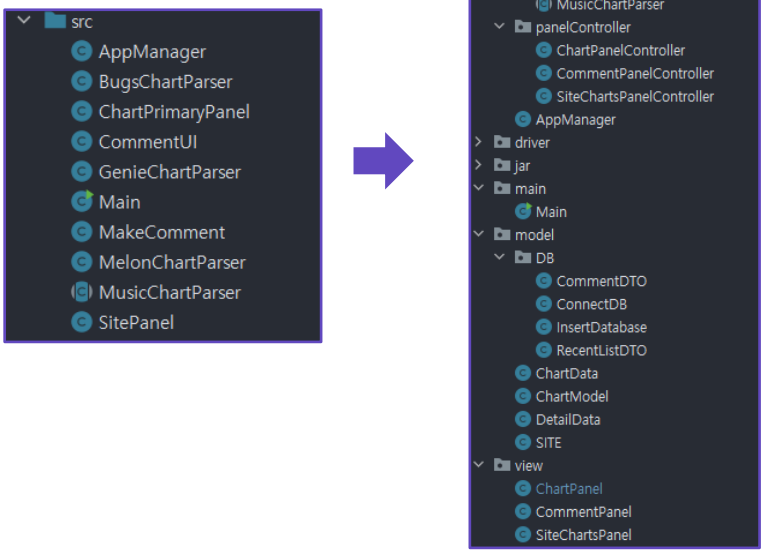
*각 사이트 : 멜론, 벅스, 지니

오리지널 프로젝트의 문제점

DB 부재 -> DB 생성 및 작성

*.txt -> DB

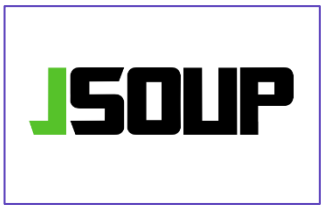
MVC 패턴 분리 필요성 ->Model / View / Controller 분리



< MVC 분리 이전 이후 클래스 변화 >

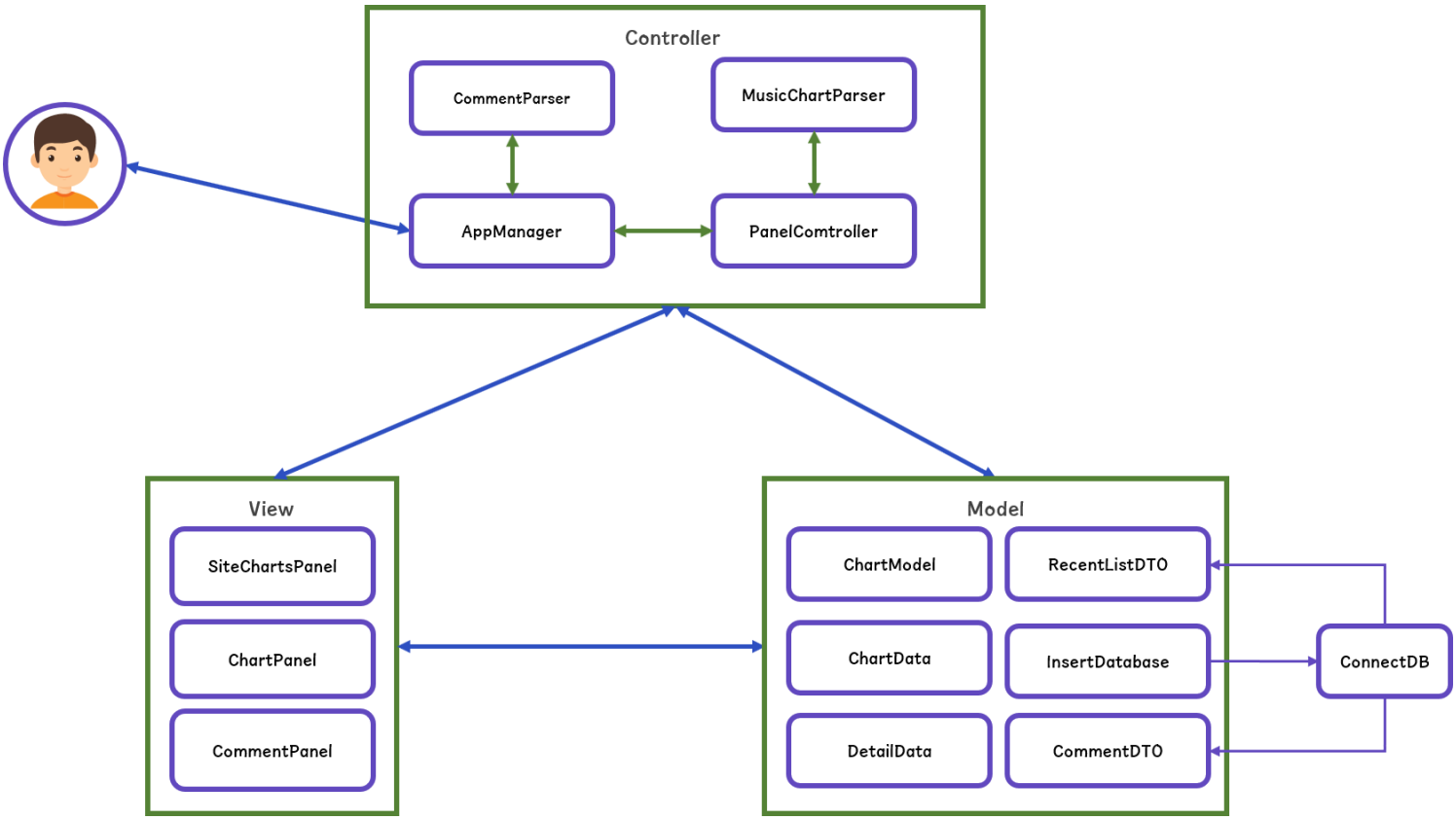
사용된 프레임워크 및 라이브러리

웹 크롤러



DB

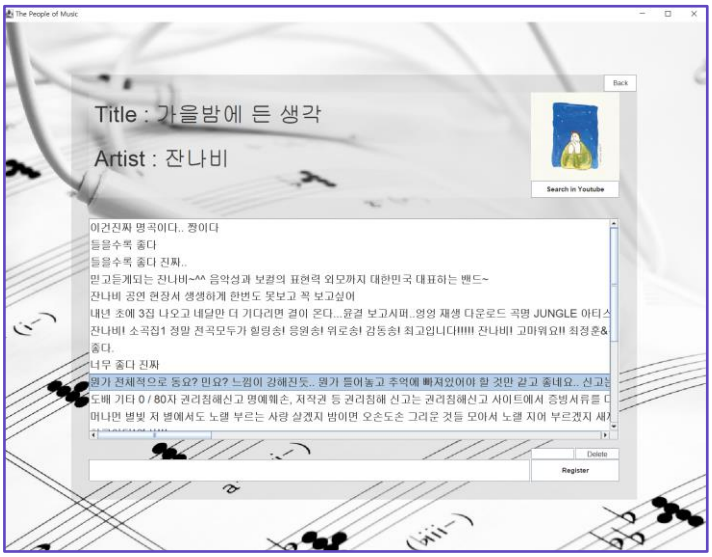
리펙토링 이후 프로젝트 구조도



< 리팩토링 이후 전체 구조도 >

프로그램 작동 사진 및 영상

웹 크롤링 영상 [링크]



< 프로젝트 시연 사진 >

최인호 리펙토링 내용

담당 파트

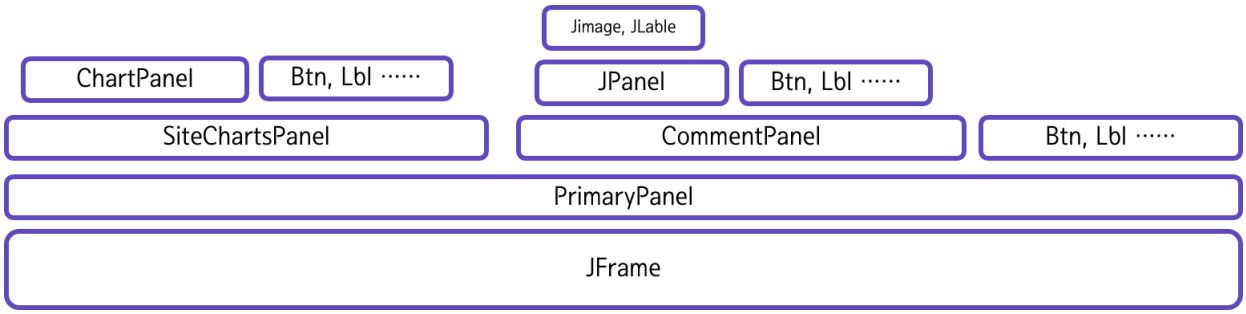
- View
- CommentParser

일정

9 주차	MVC패턴 분리View 리펙토링
10 주차	
11 주차	프로젝트 전체 조율 및 역할 분배
12 주차	CommentParser 생성
13 주차	CommentParser 리펙토링
14 주차	CodeReview 피드백 반영 및 프로젝트 전체 점검

View 리펙토링

구조도 개선



각자 선언되어 있던 Panel 클래스들을 JFrame 위에 PrimaryPanel 위로 올린 후 적층 적인 구조로 리펙토링을 진행했습니다.

1. ChartPanel

```
public SitePanel() {
    strChartName = "Melon"; //프로그램 실행 직후 Melon 차트를 표시하기 위함
    clkListener = new ClickListener();

    setBackground(LBLBACKGROUND);
    setLayout(null);
    setFont(new Font( name: "맑은 고딕", Font.BOLD, size: 64));

    lblTitle = new JLabel( text: strChartName + " TOP 100");
    lblTitle.setBackground(Color.white);
    lblTitle.setForeground(TITLECOLOR);
    lblTitle.setBounds( x: 80, y: 30, width: 920, height: 80);
    lblTitle.setFont(new Font( name: "배달의민족 도현", Font.BOLD, size: 48));
    lblTitle.setHorizontalAlignment(SwingConstants.CENTER);
    lblTitle.setVerticalAlignment(SwingConstants.CENTER);
    add(lblTitle);

    MusicChartParser parser = AppManager.getS_instance().getParser();
    if(!parser.isParsed()) parser.chartDataParsing( parentComponent: this); //

    tableModel = new ChartModel(parser.getChartList());

    tableSorter = new TableRowSorter<ChartModel>(tableModel);
    tableSorter.setComparator( column: 0, new Comparator<Integer>() {...}); //

    tableChart = new JTable(tableModel);
    tableChart.setBackground(LISTBACKGROUND);
    tableChart.setForeground(TEXTCOLOR);
    tableChart.setFont(new Font( name: "맑은 고딕", Font.PLAIN, size: 18));
    tableChart.setRowHeight(60);
    buildTable();
    tableChart.setRowSorter(tableSorter);
    tableChart.addMouseListener(clkListener);
    add(tableChart);

    scrollBar = new JScrollPane(tableChart, JScrollPane.VERTICAL_SCROLLBAR_ALW
    scrollBar.setBounds( x: 40, y: 130, width: 1000, height: 540);
    add(scrollBar);
} //생성자 끝
```

```
public ChartPanel() {
    _strChartName = "Melon"; //프로그램 실행 직후 Melon 차트를 표시하기 위함

    setBackground(_ColorLblBackground);
    setLayout(null);
    setFont(new Font( name: "맑은 고딕", Font.BOLD, size: 64));

    setInitTableChart();
    setInitScrollBar();
    setInitLblTitle();

    new ChartPanelController( theChartPanel: this);
} //생성자 끝
```

1. SitePanel -> ChartPanel 클래스 명 변경

Site 라는 이중적인 표현 대신에 정확히 Chart 를 띄워주는 클래스로 ChartPanel 로 클래스명 변경

2. constructor 내 메소드 분리

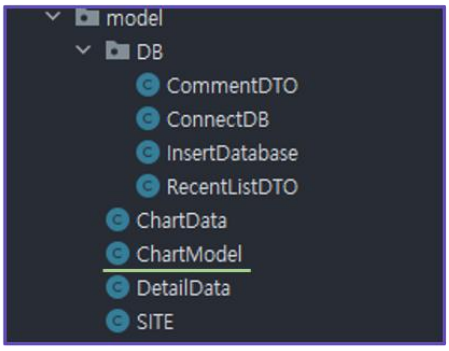
constructor 에 너무 많은 변수들이 선언되고 사용되고 있기에 메소드 분리를 진행했습니다.

```
public class ChartModel extends AbstractTableModel {
    //...
    private String[] arrColumnName;
    //각각의 셀의 항목을 저장하는 배열
    private Object[][] chartData;

    // - - - - - 생성자 - - - - -
    public ChartModel(JSONArray musics) {...} //생성자 끝

    /*...*/
    private void setContents(JSONArray musics) {...}

    public Object[][] getChartData() { return chartData; }
    public Object[] getMusicData(int index) { return chartData[index]; }
    @Override
    public int getColumnCount() { return arrColumnName.length; }
    @Override
    public int getRowCount() { return chartData.length; }
    @Override
    public Object getValueAt(int row, int column) { return chartData[row][column]; }
    @Override
    public void setValueAt(Object value, int row, int column) { chartData[row][column] = value; }
    @Override
    public String getColumnName(int column) { return arrColumnName[column]; }
    @Override
    public Class<?> getColumnClass(int column) {...}
    @Override
    public boolean isCellEditable(int rowIndex, int columnIndex) { return false; }
} //ChartModel 클래스 끝
```



3. Inner Class 분리

차트 모델이 되는 이내 클래스를 Model 파트로 클래스 분리를 진행했습니다.

4. Switch 문 제거

```
switch(AppManager.getS_instance().getSite_M_B_G()){
    case 1:
        strChartName = "Melon";
        break;
    case 2:
        strChartName = "Bugs";
        break;
    case 3:
        strChartName = "Genie";
        break;
}
```

선언

```
public static final String[] SITENAME = {"MELON","BUGS","GENIE"};
```

리팩토링 결과

```
_strChartName = SITE.SITENAME[ChartData.getS_instance().getSite_M_B_G()];
```

switch 문을 static final 선언으로 전환하여서 한 눈에 보기 쉽게 리펙토링을 진행했습니다.

5. 메소드 명 변경

```
public void filter(String text, int criteria) {...}
```

```
public void filterTitleANDArtist(String text, int criteria) {...}
```

filter 라는 메소드 명은 어떤 것을 구체적으로 filter 해주는지 명확하지 않기에 메소드 명을 변경

6. 계층적 구조 변경

```
private void setInitTableChart() {
    setInitTableModel();
    setInitTableSorter();

    _tableChart = new JTable(_tableModel);
    _tableChart.setBackground(_ColorListBackground);
    _tableChart.setForeground(_ColorTextColor);
    _tableChart.setFont(new Font( name: "맑은 고딕", Font.PLAIN,  size: 18));
    _tableChart.setRowHeight(60);
    makeTable();
    _tableChart.setRowSorter(_tableSorter);
    add(_tableChart);
}
```

```
private void setInitTableModel() {
    if (!ChartData.getS_instance().getParser().isParsed())
        ChartData.getS_instance().getParser().chartDataParsing( parentComponent: this); //

    _tableModel = new ChartModel(ChartData.getS_instance().getParser().getChartList());
}
```

TableChart위에 TableModel이 있듯 TableChart를 정의하는 메소드에서 TableModel를 선언하는 메소드를 호출하게끔 리펙토링을 했습니다.

2. SiteChartsPanel

1. ChartPrimaryPanel -> SiteChartsPanel 클래스명 변경

PrimaryPanel은 이미 존재하는 JPanel이기에 Chart를 모아둔 클래스이기에 클래스명 변경

2. Constructor 내 메소드 분리

```
public SiteChartsPanel() {
    new InsertDatabase().insertChartDatabase( parentComponent: this);

    setBackground(new Color( r: 255,  g: 255,  b: 255,  a: 0));
    setBounds( x: 1,  y: 0,  width: 1278,  height: 960);
    setLayout(null);

    setInitBtnRefresh();

    setInitStrCombo();

    setInitBtnSearch();

    setInitTextSearch();

    setInitPnlChartPanel();

    setInitBtnSites();

    setInitLblTime();

}//constructor
```

```
private void setInitBtnRefresh() {
    _btnRefresh = new JButton(new ImageIcon( filename: "Image/Refresh.png"));

    _btnRefresh.setBounds( x: 30,  y: 30,  width: 40,  height: 40);
    _btnRefresh.setForeground(Color.DARK_GRAY);
    _btnRefresh.setBackground(Color.lightGray);
    this.add(_btnRefresh);
}
```

constructor에 너무 많은 변수들이 선언되고 사용되고 있기에 메소드 분리를 진행했습니다.

3. CommentPanel

1. CommentUI -> CommentPanel

CommentUI 보다는 통일성 있는 클래스명 선언을 위해 CommentPanel로 클래스명 변경

2. 계층적 구조 변경

```
private void setInitializationPnlComment() { // Called by Constructor
    _pnlComment = new JPanel();
    _pnlComment.setBounds( x: 32,  y: 260,  width: 960,  height: 640);
    _pnlComment.setBackground(new Color( r: 0,  g: 0,  b: 0,  a: 0));
    _pnlComment.setLayout(null);
    add(_pnlComment);

    setInitializationListComment();
    setInitializationTxtComment();
    setInitializationTxtPassword();
    setInitializationBtnRegister();
    setInitializationBtnDelete();
    setInitializationScroll();
}
```

PnlComment 위에 올라가는 라벨, 리스트, 스크롤 등이 계층적으로 생성 및 호출 되게끔 변경

JPanel위에 올라가서 사용되는 것들은 해당 JPanel선언 하는 메소드에서 호출하도록 리펙토링 진행

3. 메소드 명 변경

```
private void removeAtTxt(int index){...}
/*...*/
private void clearAll(){...}
```

→

```
public void clearMusicData() {...}

public void clearPanelTxt() {...}
```

어떤 txt를 제거하는지 어떤 것을 지우는지 명확하지 않은 메소드는 메소드 명 변경을 진행했습니다.

4. CommentParser

1. 기능 추가 이유

초기 프로젝트 진행 시 댓글을 크롤링하여서 보여주고 싶었으나 어떻게 가져와야 할 지 몰라서 포기했지만, 본 프로젝트 의미에 맞는 기능이기에 추가했습니다.

2. 사용된 라이브러리 및 언어

사용 언어 : 코틀린

사용된 프레임 워크 : Selinium (자바스크립트 실행 이후 댓글을 띄워주기에 Selinium을 사용했습니다.)

사용한 라이브러리 : Jsoup

3. 리펙토링 내용

1. 공통 변수 제거

```
class CrwalCommentTest {
    //Properties
    private var driver: WebDriver? = null
    private val WEB_DRIVER_ID = "webdriver.chrome.driver"
    private val WEB_DRIVER_PATH = "src/driver/chromedriver.exe"
    val albumnum = "81743752"
    private var base_url: String? = "https://music.bugs.co.kr/album/20356321?wl_ref=list_tr_11_chart"

    init {
        System.setProperty(WEB_DRIVER_ID, WEB_DRIVER_PATH)
        driver = ChromeDriver()
    }
```

초기 구현 : 각 클래스마다 크롬드라이버를 구현해서 생성

파라미터 변수

```
class BugsAlbumCommentParser(var driver: WebDriver) {
```

선언

```
private void crawlingComments(){
    System.setProperty("webdriver.chrome.driver","src/driver/chromedriver.exe");
    WebDriver driver = new ChromeDriver();

    InsertDatabase.insertCommentDatabase(new MelonAlbumCommentParser(driver).crawl());
    InsertDatabase.insertCommentDatabase(new BugsAlbumCommentParser(driver).crawl());
    InsertDatabase.insertCommentDatabase(new GenieAlbumCommentParser(driver).crawl());

    driver.quit();
}
```

리팩토링 하면서 드라이버를 매개변수로 받아서 클래스간 공통 변수 공유

초기에는 각 클래스마다 크롬드라이버를 구현해서 생성했지만 크롬드라이버를 구동 시키는 작업은 무겁기에 리펙토링을 진행하면서 공통적인 크롬드라이버를 선언부에서 파라미터로 전달하는 형식으로 리펙토링을 진행했습니다.

2. 메소드 분리

```
private fun MutableList<String>.refine(): MutableList<String> {
    val result = mutableListOf<String>()
    var str : String
    for(i in 3 until this.size){
        str = this[i].replace( oldValue: " NEW ", newValue: "").filter { it in '가'..'힣' || it.toInt() in 0..127 }
        if(str.contains( other: "더보기 재생 다운로드 곡명"))
            result.add(str.substring(1,str.indexOf( string: "더보기 재생 다운로드 곡명")))
        else
            result.add(str.substring( startIndex: 1))
    }
    return result
}

private fun getAlbumIDtoSet(): Set<String> {
    val result = mutableSetOf<String>()
    for (i in 1..100) {
        if (!ChartData.getS_instance().melonChartParser.isParsed)
            ChartData.getS_instance().melonChartParser.chartDataParsing( parentComponent: null)
        result.add(ChartData.getS_instance().melonChartParser.getAlbumID(i).filter { it in '0'..'9' })
    }
    return result
}
```

각 차트 앨범 사이트를 구분하는 ID를 Set로 반환하는 메소드를 분리를 리펙토링하면서 같은 앨범을 파싱하는 일을 사전에 방지했습니다.

Jsoup에서 얻어온 html파일을 DB에 넣을 수 있게끔 필터링하는 메소드를 분리했습니다.

개인 소감

전에는 자신의 코드를 작성하기만 했지 되돌아 본적은 없었습니다. 이번 수업 덕분에 리펙토링을 접할 수 있어서 새로웠습니다. 코드의 작성에 답은 없지만 옳은 길은 있다는 것도 조금은 깨달은 것 같습니다. 특히 이번 프로젝트에서 리펙토링뿐만 아니라 기능추가를 함께 했습니다. 기존 프로젝트의 기능 추가와 리펙토링 후 기능추가는 달랐습니다. 기존에는 기능을 구현해 놓고 그 기능을 추가하는데 상당한 에러들이 많이 발생하였었습니다. 변수가 맞지 않거나, 기능을 추가하기 위해 다른 클래스 곳곳에 코드를 추가하니 코드가 어지러웠지만, 이번에 리펙토링 후 기능을 추가하니 어디에 코드가 있어야 하는지 알기에 기존 코드에 추가하기가 깔끔했고, 새로운 코드 작성하면서도 클린 코드를 작성하도록 노력을 하게 된다는 점이 달랐습니다. 이번 수업을 통해서 리펙토링의 이유와 중요성을 알 수 있어서 좋았습니다.

- 안건우 리펙토링 내용

담당 파트

AppManager

MusicChartParser

ChartData, DetailData

일정

9 주차	AppManager 리펙토링 및 MVC 분리
10 주차	
11 주차	카멜 표기법
12 주차	MusicChartParser 리펙토링
13 주차	ChartData, DetailData 리펙토링
14 주차	CodeReview 피드백 반영

AppManager 리펙토링

```
public String getTitle(int rank) {...} // String getTitle(int rank)

public String getTitle(JSONObject jObj) {...} // String getTitle(JSONObject jObj)

public String getArtistName(int rank) {...} // String getArtistName(int rank)

public String getArtistName(String title) {...} // String getArtistName(String title)

public String getArtistName(JSONObject jObj) {...} // String getArtistName(JSONObject jObj)

public String getAlbumName(int rank) {...} // String getArtistName(int rank)

public String getAlbumName(String title) {...} // String getAlbumName(String title)

public String getAlbumName(JSONObject jObj) {...} // String getAlbumName(JSONObject jObj)

public String getSongId(int rank) {...} // String getSongId(int rank)

public String getSongId(String title) {...} // String getSongId(String title)

public String getSongId(JSONObject jObj) {...} // String getSongId(JSONObject jObj)
```

<메서드 이동 전 AppManager>

AppManager 의 메서드가 MVC 의 기능을 전부 수행하여 메서드 이동 진행

<AppManager 메서드 이동>

AppManager의 메서드들을 각각의 MVC Class로 이동하였다.

표기법

표기법을 카멜 표기법으로 통일하였다.

```
public MelonChartParser() { //
    _songCount = 0;
    _chartList = null;
    _songDetailInfo = null;
    _url = null;
    _chartThread = null;
    _songDetailThread = null;
    progressMonitor = null;
} // constructor
```

<카멜 표기법 적용>

MusicChartParser 리펙토링 (피드백 반영)

MusicChartParser에는 여러 메서드 안에 중복된 코드가 존재한다.

중복된 코드를 메서드로 추출하였다.

```
public String getSongId(int rank) { // 노래 순위를 통해 해당 노래의 앨범 이름을 반환하는 메소드
    if (rank < 1 || rank > 100) { // 1 <= rank <= 100을 벗어나는 범위라면
        System.out.println("1~100위 이내의 순위를 입력해주세요");
        return null;
    }

    if (!isParsed()) { // 파싱이 이루어지지 않았다면
        System.out.println(isNotParsed);
        return null;
    }

    if (songCount == 1) { // 노래 한 곡에 대한 상세 파싱이 이루어졌다면
        System.out.println("getSongId(int rank) : " + isOnlyChartParse);
        return null;
    }

    return ((JSONObject) chartList.get(rank - 1)).get("songId").toString();
} // String getSongId(int rank)
```

<매개변수가 rank인 메서드>

```java
public String getAlbumName(int rank) { // 노래 순위를 통해 해당 노래의 앨범 이름을 반환
    if (rank < 1 || rank > 100) { // 1 <= rank <= 100을 벗어나는 범위라면
        System.out.println("1~100위 이내의 순위를 입력해주세요");
        return null;
    }
    if (!isParsed()) { // 파싱이 이루어지지 않았다면
        System.out.println(isNotParsed);
        return null;
    }

    if (songCount == 1) { // 노래 한 곡에 대한 상세 파싱이 이루어졌다면
        System.out.println("getAlbumName(int rank) : " + isOnlyChartParse);
        return null;
    }

    return ((JSONObject) chartList.get(rank - 1)).get("albumName").toString();
} // String getArtistName(int rank)
```

<매개변수가 rank인 메서드>

공통된 rank 값을 검사하는 코드를 rank를 매개변수로 받는 메서드로 추출하였다.

```java
boolean isRanked(int rank){
    if (rank < 1 || rank > 100) { // 1 <= rank <= 100을 벗어나는 범위라면
        System.out.println("1~100위 이내의 순위를 입력해주세요");
        return false;
    }
    return true;
}
```

<Rank 값 검사 코드의 메서드 추출>

Parsing 여부를 검사하는 코드를 메서드로 추출하였다.

```java
boolean isSongDetailParsed(){
    if (!isParsed()) { // 파싱이 이루어지지 않았다면
        System.out.println(_isNotParsed);
        return false;
    }
    if (_songCount == 1){// 노래 한 곡에 대한 상세 파싱
        System.out.println(_isOnlyChartParse);
        return false;
    }
    return true;
}
```

<Parsing 여부를 검사하는 코드의 메서드 추출>

```
public String getSongId(String title) { // 노래 제목을 통해 해당 노래의 노래 아이디를
    if (!isParsed()) { // 파싱이 이루어지지 않았다면
        System.out.println(isNotParsed);
        return null;
    }

    if (songCount == 1) { // 노래  한 곡에 대한 상세 파싱이 이루어졌다면
        System.out.println("getSongId(String title) : " + isOnlyChartParse);
        return null;
    }

    for (int i = 0; i < songCount; i++) { // 차트 100곡에 대한 파싱이 이루어졌다면 JS
        if (((JSONObject) chartList.get(i)).get("title") == title)
            return ((JSONObject) chartList.get(i)).get("songId").toString();
    }
    return null;
} // String getSongId(String title)
```

<매개변수가 title인 메서드>

```
public String getAlbumName(String title) { // 노래 제목을 통해 해당 노래의 앨범 이름을
    if (!isParsed()) { // 파싱이 이루어지지 않았다면
        System.out.println(isNotParsed);
        return null;
    }

    if (songCount == 1) { // 노래 한 곡에 대한 상세 파싱이 이루어졌다면
        System.out.println("getAlbumName(String title) : " + isOnlyChartParse);
        return null;
    }

    for (int i = 0; i < songCount; i++) { // 차트 100곡에 대한 파싱이 이루어졌다면 JSO
        if (((JSONObject) chartList.get(i)).get("title") == title)
            return ((JSONObject) chartList.get(i)).get("albumName").toString();
    }
    return null;
} // String getAlbumName(String title)
```

<매개변수가 title인 메서드>

title과 일치하는 JSONObject를 찾는 코드를 title을 매개변수로 받는 메서드로 추출하였다.

```
String foundByTitle(String title, String string){
    for (int i = 0; i < _songCount; i++) { // 차트 100곡에 대한 파싱이 이루어졌다
        if (((JSONObject) _chartList.get(i)).get("title") == title)
            return ((JSONObject) _chartList.get(i)).get(string).toString();
    }
    return null;
}
```

<title과 일치하는 JSONObject를 찾는 코드의 메서드 추출>

Parsing 코드를 메서드로 추출하였다.

```java
@Override
public void songDetailDataParsing(JSONObject jObj, Component parentComponent) { // 노래 한 곡
    if (jObj == null) {
        System.out.println(plzUseRightJSONObject);
        return;
    }

    if (!jObj.containsKey("songId")) { // songId key값 유효성 검사
        System.out.println(jsonDontHaveKey);
        return;
    }
    url = "https://www.melon.com/song/detail.htm?songId=" + jObj.get("songId").toString();
    if (songDetailThread != null) { // Thread를 사용하는 게 처음이 아닐 때
        if (songDetailThread.isAlive()) // Thread가 살아있으면 정지
            songDetailThread.stop();
    }
    songDetailThread = new Thread(new SongDetailDataParsingThread()); // Thread는 재사용이 안
    songDetailThread.start();
    try {
        songDetailThread.join(); // SongDetailDataParsingThread가 종료되기 전까지 대기
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
} // songDetailDataParsing(JSONObject jObj, Component parentComponent)
```

<JSONObject를 매개변수로 하는 메서드>

```java
@Override
public void songDetailDataParsing(int rank, JSONArray chartListData, Component parentComponent) {
    if (chartListData == null) {
        System.out.println("차트 파싱된 데이터가 없어 메소드 실행을 종료합니다 :(");
        return;
    }
    url = "https://www.melon.com/song/detail.htm?songId="
            + ((JSONObject) chartListData.get(rank - 1)).get("songId").toString(); // 파싱할 url을 만

    if (songDetailThread != null) { // Thread를 사용하는 게 처음이 아닐 때
        if (songDetailThread.isAlive()) // Thread가 살아있으면 정지
            songDetailThread.stop();
    }
    songDetailThread = new Thread(new SongDetailDataParsingThread()); // Thread는 재사용이 안되기 때문
    songDetailThread.start();
    try {
        songDetailThread.join(); // SongDetailDataParsingThread가 종료되기 전까지 대기
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
} // songDetailDataParsing(int rank, JSONArray chartListData, Component parentComponent)
```

<rank와 JSONArray를 매개변수로 하는 메서드>

ChartData, DetailData 리펙토링

ChartData와 DetailData 클래스는 분류 부호 _siteMBG를 사용한다.

분류를 위한 switch 문 하나만 남긴 후 나머지 switch 문을 수정하였다.

```
public MusicChartParser getParser() {
    switch (_siteMBG) {
        case SITE.MELON:
            return melon;
        case SITE.BUGS:
            return bugs;
        case SITE.GENIE:
            return genie;
        default:
            return null;
    }
}
```

<분류를 위한 switch 문>

```
public void DataPassing(Component parentComponent) {
    System.out.println("Parsing Data");
    switch (Site_M_B_G) {
        case 1:
            melon.chartDataParsing(parentComponent);
            break;
        case 2:
            bugs.chartDataParsing(parentComponent);
            break;
        case 3:
            genie.chartDataParsing(parentComponent);
            break;
    }
}
```

<switch 문 리펙토링 전 DataPassing 메서드>

```
public void DataPassing(Component parentComponent) {
    System.out.println("Parsing Data");
    getParser().chartDataParsing(parentComponent);
}
```

<switch 문 리펙토링 후 DataPassing 메서드>

```java
public void detailDataPassing(int rank, JSONArray chartListData, Component parentComponent){
    System.out.println("Parsing Detail Data");
    switch(Site_M_B_G){
        case 1:
            if(detailMelon == null) detailMelon = new MelonChartParser();
            detailMelon.songDetailDataParsing(rank, chartListData, parentComponent);
            //Parsing Method
            break;
        case 2:
            if(detailBugs == null)
                detailBugs = new BugsChartParser();
            detailBugs.songDetailDataParsing(rank, chartListData, parentComponent);
            System.out.println("Bugs Detail Parse");
            //Parsing Method
            break;
        case 3:
            if(detailGenie == null)
                detailGenie = new GenieChartParser();
            detailGenie.songDetailDataParsing(rank, chartListData, parentComponent);
            System.out.println("Genie Detail Parse");
            //Parsing Method
            break;
    }
}
```

<switch 문 리펙토링 전 detailDataPassing 메서드>

```java
public void detailDataPassing(int rank, JSONArray chartListData, Component parentComponent){
    System.out.println("Parsing Detail Data");
    getParser().songDetailDataParsing(((JSONObject) chartListData.get(rank - 1)), parentComponent);
}
```

<Switch 문 리펙토링 후 detailDataPassing 메서드>

개인소감

메서드 추출, 메서드 이동 등 여러 리펙토링 기법을 통해 효율적이고 보기 좋게 코딩할 수 있는 법을 알게 되었고 앞으로도 중간중간 리펙토링을 해야 함을 느꼈다.

- 박기춘 리펙토링 내용

담당 파트

전반적인 DB관련 model파트 및 관련된 컨트롤러의 일부를 담당하였습니다.

일정

9 주차	로컬 DB에서 AWS DB로 연결
10 주차	코멘트 리펙토링 - 메서드 분리, 추출, 변수
11 주차	
12 주차	DAO, DTO 분리 및 데이터베이스 리펙토링
13 주차	최근 본 목록 기능 추가 및 리펙토링
14 주차	코드 리뷰 피드백 반영 및 미흡한 부분 추가 수정

Data Base

localhost 에 연결된 DB 를 AWS 를 이용하여 외부에서 별도의 환경설정 없이 접근이 가능하도록 했습니다.

```
String url = "jdbc:mysql://localhost:3306/charts_db?&serverTimezone=Asia/Seoul&useSSL=false";
String userid = "root";
String pwd = "root";
// 1.드라이버 로딩
try {
    Class.forName("com.mysql.jdbc.Driver");
    System.out.println("드라이버 로드 성공");
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}

// 2.연결
try {
    System.out.println("데이터베이스 연결 준비...");
    con = DriverManager.getConnection(url, userid, pwd);
    System.out.println("데이터베이스 연결 성공");
} catch (SQLException e) {
    e.printStackTrace();
}
```

<변경 전 - localhost에 연결된 데이터베이스>

```
public void connectionDB() {
    String url = "jdbc:mysql://database-2.cqcrpm8zqs1t.ap-northeast-2.rds.amazonaws.com:3306/charts_db?&serverTimezone=Asia/Seoul&useSSL=false";
    String userid = "admin";
    String pwd = "refactoring";

    // 2.연결
    try {
        System.out.println("데이터베이스 연결 준비...");
        _con = DriverManager.getConnection(url, userid, pwd);
        System.out.println("데이터베이스 연결 성공");
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

<변경 후 - AWS에 연결된 데이터베이스>

표기법을 카멜 표기법으로 통일하였다.

```
private PreparedStatement _pstmt = null;
private ResultSet _rs = null;
private Statement _stmt = null;
private Connection _con = null;
private String _sql;

final private String _url = "jdbc:mysql://database-2.cg
final private String _userId = "admin";
final private String _password = "refactoring";
```

<카멜 표기법 적용>

하나의 메서드 안에서 여러가지 기능을 처리하였다.

```
public void actionPerformed(ActionEvent e) {
    if (!theCommentPanel.txtComment.getText().equals("")) {
        con = ConnectDB.GetDB();
        try {
            String sql = "INSERT INTO songinfo VALUES (?, ?, ?, ?, ?, ?)";
            pstmt = con.prepareStatement(sql);
            pstmt.setString(1, theCommentPanel.sqltitle);
            pstmt.setString(2, theCommentPanel.strArtist);
            pstmt.setString(3, AppManager.getS_instance().getParser().getAlbumName(theCommentPanel.strTitle));
            pstmt.setInt(4, AppManager.getS_instance().getSite_M_B_G());
            pstmt.setString(5, theCommentPanel.txtComment.getText());
            pstmt.setString(6, theCommentPanel.txtPassword.getText());
            pstmt.executeUpdate();
            theCommentPanel.modelList.addElement(theCommentPanel.txtComment.getText());

            theCommentPanel.arrComment.add(theCommentPanel.txtComment.getText());
            if (theCommentPanel.txtPassword.getText().equals(""))
                theCommentPanel.arrPassword.add("0000");
            else
                theCommentPanel.arrPassword.add(theCommentPanel.txtPassword.getText());
            theCommentPanel.txtComment.setText("");
            theCommentPanel.txtPassword.setText("");
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
    }//obj == btnRegister
}//actionPerfomed
```

<변경 전 - 여러 기능이 같이 있는 하나의 메서드>

```java
public void actionPerformed(ActionEvent e) {
    if (!the_Comment_Panel._txtComment.getText().equals("")) {
        DB.connectionDB();
        DB.insertCommentDB(the_Comment_Panel._strAlbumId,  order: 6,
                the_Comment_Panel._txtComment.getText(),
                the_Comment_Panel._txtPassword.getText());

        the_Comment_Panel._modelList.addElement(the_Comment_Panel._txtComment.getText());

        CommentDTO list = new CommentDTO();
        list.setComment(the_Comment_Panel._txtComment.getText());

        if (the_Comment_Panel._txtPassword.getText().equals(""))
            list.setPassword("0000");
        else
            list.setPassword(the_Comment_Panel._txtPassword.getText());

        the_Comment_Panel._commentListDTO.add(list);
        the_Comment_Panel.clearPanelTxt();
    }//obj == btnRegister
}//actionPerformed
```

```java
public void insertCommentDB(String albumID, int order, String comment, String pwd){
    try {
        _sql = "INSERT INTO commentList VALUES (?, ?, ?, ?)";
        _pstmt = _con.prepareStatement(_sql);
        _pstmt.setString( parameterIndex: 1, albumID);
        _pstmt.setInt( parameterIndex: 2, order);
        _pstmt.setString( parameterIndex: 3, comment);
        _pstmt.setString( parameterIndex: 4, pwd);
        _pstmt.executeUpdate();
    } catch (SQLException e1) {
        e1.printStackTrace();
    }
}
```

<변경 후 - 메서드 추출 및 클래스 분리>

DAO, DTO 분리를 통한 반복 사용되는 코드의 제거

```java
public ArrayList<String> _arrComment;
public ArrayList<String> _arrPassword;
```

```java
DB.getDB();
ArrayList<String> albumIdList = DB.getAlbumId(_sqlTitle);
System.out.println("albumIdList : "+albumIdList);
for (String albumId : albumIdList) {
    addArrayListString(_arrComment, DB.readCommentDB(albumId));
    addArrayListString(_arrPassword, DB.readPwdDB(albumId));
    System.out.println("comment " + _arrComment + ", pwd" + _arrPassword);
}
```

```java
public ArrayList<String> readCommentDB(String albumId){
    ArrayList<String> comment = new ArrayList<String>();
    try {
        _stmt = _con.createStatement();
        _sql = "SELECT comment FROM commentList WHERE albumId = '" + albumId + "'";
        _rs = _stmt.executeQuery(_sql);
        while (_rs.next()) {
            comment.add(_rs.getString("comment"));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return comment;
}
public ArrayList<String> readPwdDB(String albumId){
    ArrayList<String> password = new ArrayList<String>();
    try {
        _stmt = _con.createStatement();
        _sql = "SELECT pwd FROM commentList WHERE albumId = '" + albumId + "'";
        _rs = _stmt.executeQuery(_sql);
        while (_rs.next()) {
            password.add(_rs.getString("pwd"));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return password;
}
```

<변경 전 - ArrayList의 반복적인 사용>

```java
public class CommentDTO {
    private String _comment;
    private String _password;

    public String getComment() { return _comment; }
    public String getPassword() { return _password; }
    public void setComment(String comment) { this._comment = comment; }
    public void setPassword(String password) { this._password = password; }
}
```

```java
public ArrayList<CommentDTO> readCommentDB(ArrayList<String> albumIdList){
    ArrayList<CommentDTO> commentDTO = new ArrayList<~>();
    for (String albumId : albumIdList) {
        try {
            _stmt = _con.createStatement();
            _sql = "SELECT * FROM commentList WHERE albumId = '" + albumId + "'";
            _rs = _stmt.executeQuery(_sql);
            while (_rs.next()) {
                CommentDTO list = new CommentDTO();
                list.setComment(_rs.getString( columnLabel: "comment"));
                list.setPassword(_rs.getString( columnLabel: "pwd"));
                commentDTO.add(list);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    return commentDTO;
}
```

<변경 후 - 하나의 클래스 생성 후 DTO, DAO 분리>

랜덤으로 생성한 댓글의 저장에서 실제 크롤링한 댓글의 저장으로 변경하였다.

```
rndNum2 = (int) (Math.random() * 6) + 1;
switch (rndNum2 % 10) {
    case 1: {
        comment = singer + "는 역시 믿고 들어야지";
        break;
    }//case 1
    case 2: {
        comment = "이번 " + title + " 너무 좋은 곡 같아요";
        break;
    }//case 2
    case 3: {
        comment = "이번 앨범 " + albumName + " 너무 좋아요";
        break;
    }
    case 4: {
        comment = "앨범 발매일만 기다렸습니다!";
        break;
    }
    case 5: {
        comment = "5252~" + singer + " 기다렸다구";
        break;
    }
    case 6: {
        comment = "'" + albumName + "'앨범 수록된 노래 다 좋은 곡 같아요.";
        break;
    }
}//switch
```

```
Random rand = new Random();
passwd = "";
for (int p = 0; p < 4; p++) {
    //0~9 까지 난수 생성
    String ran = Integer.toString(rand.nextInt(10));
    passwd += ran;
}//4자리 비밀번호 설정
try {
    sql = "INSERT INTO songinfo VALUES (?, ?, ?, ?, ?, ?)";
    pstmt = con.prepareStatement(sql);
    pstmt.setString(1, title);
    pstmt.setString(2, singer);
    pstmt.setString(3, albumName);
    pstmt.setInt(4, j);
    pstmt.setString(5, comment);
    pstmt.setString(6, passwd);
    pstmt.executeUpdate();
} catch (Exception e1) {
    System.out.println("쿼리 읽기 실패 :" + e1);
}
```

<변경 전 - 랜덤으로 만들어진 댓글을 DB에 저장>

```
InsertDatabase.insertCommentDatabase(new MelonAlbumCommentParser(driver).crawl());
InsertDatabase.insertCommentDatabase(new BugsAlbumCommentParser(driver).crawl());
InsertDatabase.insertCommentDatabase(new GenieAlbumCommentParser(driver).crawl());
```

```
public void insertCommentDatabase(Map<String, List<String>> albumAndComment){
    DB.connectionDB();
    for (String albumId : albumAndComment.keySet()){
        int order = 0;
        for (String comment : albumAndComment.get(albumId)){
            //System.out.println("key : " + key +" / value : " + str);
            try {
                order++;
                if(DB.getCommentInfo(albumId).next() && order == 1){
                    //댓글이 이미 저장되어있다
                    DB.deleteCommentDB(albumId);//삭제
                    DB.insertCommentDB(albumId, order, comment, makePassword());
                }//최신 댓글들만 저장하기 위한 방법
                else{
                    DB.insertCommentDB(albumId, order, comment, makePassword());
                }
            } catch (SQLException throwables) {
                throwables.printStackTrace();
            }
        }
    }
}//크롤링한 댓글들 저장

private String makePassword(){
    Random rand = new Random();
    String passwd = "";
    for (int p = 0; p < 4; p++) {
        //0~9 까지 난수 생성
        String ran = Integer.toString(rand.nextInt( bound: 10));
        passwd += ran;
    }//4자리 비밀번호 설정
    return passwd;
}
```

```java
public void insertCommentDB(String albumID, int order, String comment, String pwd){
    try {
        _sql = "INSERT INTO commentList VALUES (?, ?, ?, ?)";
        _pstmt = _con.prepareStatement(_sql);
        _pstmt.setString( parameterIndex: 1, albumID);
        _pstmt.setInt( parameterIndex: 2, order);
        _pstmt.setString( parameterIndex: 3, comment);
        _pstmt.setString( parameterIndex: 4, pwd);
        _pstmt.executeUpdate();
    } catch (SQLException e1) {
        e1.printStackTrace();
    }
}
```

<변경 후 - 크롤링한 실제 댓글을 DB에 저장하는 메서드를 만들고 분리>

개인소감

리펙토링 기법을 통해 길어졌던 코드를 간단하게 만들면서 예전에 이 프로젝트를 진행할 때 얼마나 코드가 비효율적이었는지 알게 되었다. 그리고 새로운 기능을 만들고 리펙토링하면서 설계의 중요성을 다시 한번 깨닫게 되는 계기가 되었다.

- **김성수 리펙토링 내용**

담당 파트

MVC Controller 분리

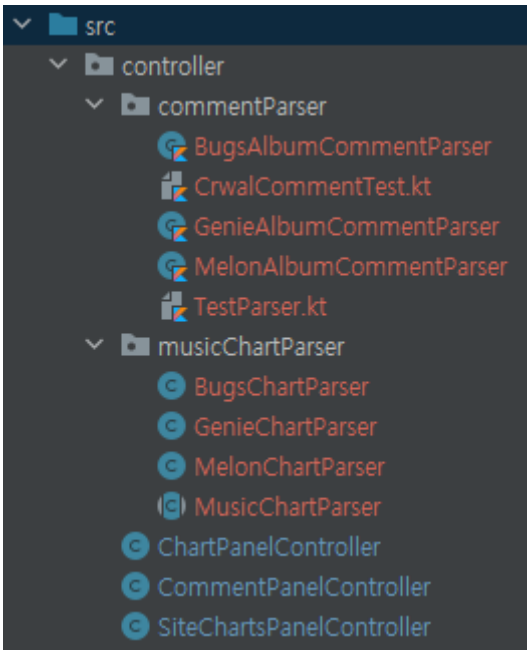
최근 본 음악 목록 -> YouTube 음악 검색

13주차 까지의 일정

9 주차	MVC 분리 마무리
10 주차	스위치문 정리
11 주차	메서드 정리
12 주차	유튜브 음악 검색 기능추가
13 주차	추가 기능 마무리
14 주차	코드리뷰 피드백

MVC 분리 및 코드 리펙토링

MVC 모델 Controller 분리



기존 코드들이 MVC 분리 없이 한 클래스 내부에 전부 존재하였음

분리 후 컨트롤러를 패키지로 묶고 리펙토링 중 생성된 Parser들 또한 컨트롤러로 이동

```java
public class ChartPanelController {

    private ChartPanel theChartPanel;

    public ChartPanelController(ChartPanel TheChartPanel) {...}

    private class addClickListener implements MouseListener {...} //addClickListener

    public void PopUpCommentUI(int rank){...}
}
```

```java
public class CommentPanelController {

    ConnectDB DB = new ConnectDB();

    private final CommentPanel theCommentPanel;

    public CommentPanelController(CommentPanel TheCommentPanel) {...}

    private class ButtonRegisterListener implements ActionListener{...}//ButtonRegisterListener

    private class ButtonDeleteListener implements ActionListener{...}//ButtonDeleteListener

    private class ButtonBackListener implements ActionListener{...}//ButtonBackListener

    private class ButtonYouTubeListener implements ActionListener{...}//ButtonYoutubeListener

    public void BackToChartPrimaryPanel(){...}
}
```

```java
public class SiteChartsPanelController {

    private SiteChartsPanel theSiteChartsPanel;

    public SiteChartsPanelController(SiteChartsPanel TheSiteChartsPanel) {...}

    private class ButtonRefreshListener implements ActionListener {...}//ButtonRefreshListener

    public void setChartTime() {...}

    public String getChartTime() {...}

    private void renewalChartData() {...}

    private class ButtonMelonListener implements ActionListener {...}//ButtonMelonListener

    private class ButtonBugsListener implements ActionListener {...}//ButtonBugsListener

    private class ButtonGenieListener implements ActionListener {...}//ButtonGenieListener

    private class ButtonRecentListener implements ActionListener {...}//ButtonRecentListener

    private class KeyActionListener implements KeyListener{...}//KeyListener
}
```

컨트롤러 클래스들은 상속된 패널의 컨트롤러 기능과 컨트롤러 기능에 포함된 메소드로 구성

코드 리펙토링

스위치문 정리

변경 전

```
    public void actionPerformed(ActionEvent e) {
        switch (ChartData.getS_instance().getSite_M_B_G()){
            case 1:
                the_Chart_Primary_Panel._formatted_Melon = LocalDateTime.now().format(DateTimeFormatter.ofPattern("HH:mm:ss"));
                the_Chart_Primary_Panel._lblTime.setText("Renewal time : " + the_Chart_Primary_Panel._formatted_Melon);
                //   ChartData.getS_instance().setSite_M_B_G(1);
                ChartData.getS_instance().DataPassing(view_Loading);
                System.out.println("why?");
                break;
            case 2:
                the_Chart_Primary_Panel._formatted_Bugs = LocalDateTime.now().format(DateTimeFormatter.ofPattern("HH:mm:ss"));
                the_Chart_Primary_Panel._lblTime.setText("Renewal time : " + the_Chart_Primary_Panel._formatted_Bugs);
                //  ChartData.getS_instance().setSite_M_B_G(2);
                ChartData.getS_instance().DataPassing(view_Loading);
                break;
            case 3:
                the_Chart_Primary_Panel._formatted_Genie = LocalDateTime.now().format(DateTimeFormatter.ofPattern("HH:mm:ss"));
                the_Chart_Primary_Panel._lblTime.setText("Renewal time : " + the_Chart_Primary_Panel._formatted_Genie);
                // ChartData.getS_instance().setSite_M_B_G(3);
                ChartData.getS_instance().DataPassing(view_Loading);
                break;
        }
    }
}//ButtonRefreshListener
```

변경 후

```
private class ButtonRefreshListener implements ActionListener {
    private Component viewLoading;
    public ButtonRefreshListener() { }
    public ButtonRefreshListener(Component parentComponent) { viewLoading = parentComponent; }

    @Override
    public void actionPerformed(ActionEvent e) {
        setChartTime();
        theSiteChartsPanel._lblTime.setText("Renewal time : " + getChartTime());
        ChartData.getS_instance().DataPassing(viewLoading);
    }
}//ButtonRefreshListener

public void setChartTime() {
    switch (ChartData.getS_instance().getSiteMBG()) {
        case 1:
            theSiteChartsPanel._formatted_Melon = LocalDateTime.now().format(DateTimeFormatter.ofPattern("HH:mm:ss"));
            break;
        case 2:
            theSiteChartsPanel._formatted_Bugs = LocalDateTime.now().format(DateTimeFormatter.ofPattern("HH:mm:ss"));
            break;
        case 3:
            theSiteChartsPanel._formatted_Genie = LocalDateTime.now().format(DateTimeFormatter.ofPattern("HH:mm:ss"));
            break;
    }
}

public String getChartTime() {
    switch (ChartData.getS_instance().getSiteMBG()) {
        case 1:
            return theSiteChartsPanel._formatted_Melon;
        case 2:
            return theSiteChartsPanel._formatted_Bugs;
        case 3:
            return theSiteChartsPanel._formatted_Genie;
        default:
            return null;
    }
}
```

스위치문 내부에서 변수만 다른 채 공통적으로 사용되는 코드를 스위치 외부로 분리함

유사 메소드 정리

변경 전

```
private class ButtonMelonListener implements ActionListener {
    private Component view_Loading;
    public ButtonMelonListener() { }
    public ButtonMelonListener(Component parentComponent) { view_Loading = parentComponent; }

    @Override
    public void actionPerformed(ActionEvent e) {

        if(ChartData.getS_instance().getSite_M_B_G() == 1) return;
        ChartData.getS_instance().setSite_M_B_G(1);
        if(!ChartData.getS_instance().getParser().isParsed()) ChartData.getS_instance().DataPassing(view_Loading);
        System.out.println("Melon");
        the_Chart_Primary_Panel._pnlChartPanel.changeData();
        the_Chart_Primary_Panel._lblTime.setText("Renewal time : " + the_Chart_Primary_Panel._formatted_Melon);
        the_Chart_Primary_Panel._txtSearch.setText("");
        the_Chart_Primary_Panel._pnlChartPanel.filterTitleANDArtist( text: null, criteria: 2);
    }
}//ButtonMelonListener

private class ButtonBugsListener implements ActionListener {
    private Component view_Loading;
    public ButtonBugsListener() { }
    public ButtonBugsListener(Component parentComponent) { view_Loading = parentComponent; }

    @Override
    public void actionPerformed(ActionEvent e) {

        if(ChartData.getS_instance().getSite_M_B_G() == 2) return;
        ChartData.getS_instance().setSite_M_B_G(2);
        if(!ChartData.getS_instance().getParser().isParsed()) ChartData.getS_instance().DataPassing(view_Loading);
        System.out.println("Bugs");
        the_Chart_Primary_Panel._pnlChartPanel.changeData();
        the_Chart_Primary_Panel._lblTime.setText("Renewal time : " + the_Chart_Primary_Panel._formatted_Bugs);
        the_Chart_Primary_Panel._txtSearch.setText("");
        the_Chart_Primary_Panel._pnlChartPanel.filterTitleANDArtist( text: null, criteria: 2);
    }
}//ButtonBugsListener
```

변경 후

```
public String getChartTime() {
    switch (ChartData.getS_instance().getSiteMBG()) {
        case 1:
            return theSiteChartsPanel._formatted_Melon;
        case 2:
            return theSiteChartsPanel._formatted_Bugs;
        case 3:
            return theSiteChartsPanel._formatted_Genie;
        default:
            return null;
    }
}

private void renewalChartData() {
    theSiteChartsPanel._pnlChartPanel.changeData();
    theSiteChartsPanel._lblTime.setText("Renewal time : " + getChartTime());
    theSiteChartsPanel._txtSearch.setText("");
    theSiteChartsPanel._pnlChartPanel.filterTitleANDArtist( text: null, criteria: 2);
}

private class ButtonMelonListener implements ActionListener {
    private Component viewLoading;
    public ButtonMelonListener() { }
    public ButtonMelonListener(Component parentComponent) { viewLoading = parentComponent; }

    @Override
    public void actionPerformed(ActionEvent e) {

        if(ChartData.getS_instance().getSiteMBG() == 1) return;
        ChartData.getS_instance().setSiteMBG(1);
        if(!ChartData.getS_instance().getParser().isParsed()) ChartData.getS_instance().DataPassing(viewLoading);
        //System.out.println("Melon");
        renewalChartData();
    }
}//ButtonMelonListener

private class ButtonBugsListener implements ActionListener {...}//ButtonBugsListener

private class ButtonGenieListener implements ActionListener {...}//ButtonGenieListener
```

메소드에서 공통적으로 사용하는 코드를 분리했습니다.

스위치문에서도 사용되는 변수를 따로 분리한 메소드를 사용했습니다.

기능 추가 및 개선

애플리케이션 내부 곡 정보에서 “유튜브에서 검색하기” 기능을 통해 웹 팝업으로 검색 결과를 띄우는 기능을 추가

```
private class ButtonYouTubeListener implements ActionListener{
    private Component view_Loading;
    public ButtonYouTubeListener() { }
    public ButtonYouTubeListener(Component parentComponent) { view_Loading = parentComponent; }

    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            Desktop.getDesktop().browse(new URI( str: "https://www.youtube.com/results?search_query=" + the_Comment_Panel._strTitle));
        } catch (IOException ex) {
            ex.printStackTrace();
        } catch (URISyntaxException ex) {
            ex.printStackTrace();
        }
        System.out.println("Search in Youtube");
    }//actionPerformed
}//ButtonYoutubeListener
```

곡 이름에 공백 기호 및 한글 존재할 경우 인코딩 오류 발생

```
니다'ffff], pwd[5400, 9692, 8198, 3433]
ex 47: https://www.youtube.com/results?search_query=힘든 건 사랑이 아니다
```

```
URLEncoder.encode(the_Comment_Panel._strTitle,  enc: "UTF-8")))
```

기존 스트링 전체를 주소창에 입력하는 방식에서 UTF-8 형식으로 인코딩하여 입력하는 방식으로 변경함

개인 소감

MVC 분리에 대해서 더 잘 알게 되었고 세 부분을 분리하는 법에 대해 알 수 있었다. 리펙토링 시 좀 더 효율적으로 리펙토링을 하는 방법에 대해 찾아볼 수 있게 되었고. 추가 기능을 통해 인코딩의 필요성을 알게 되었다.