2020 년 2 학기

문제해결기법 코드분석

실력만은 20 학번

3 조

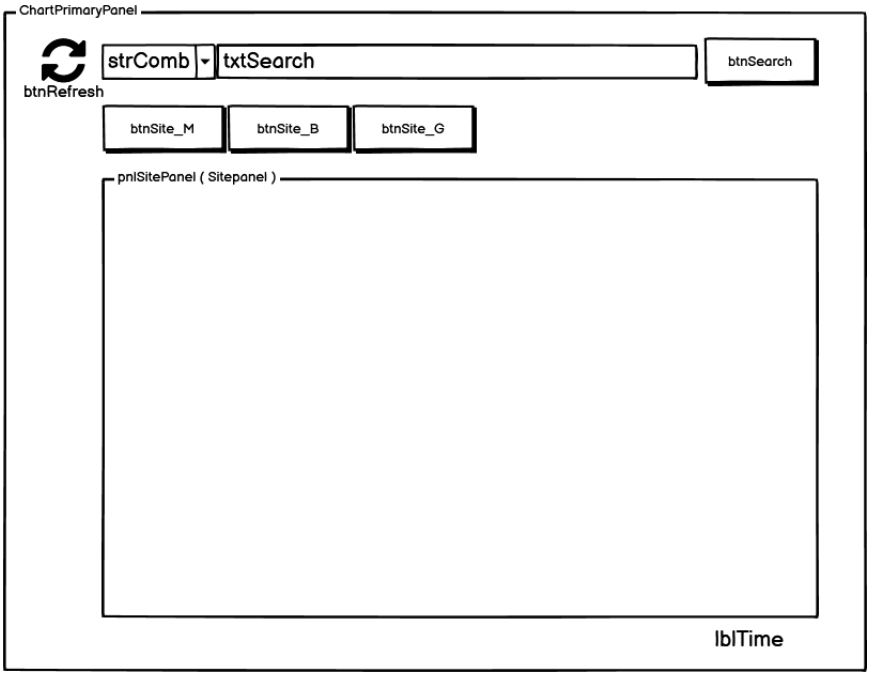
팀장 : 최인호

팀원 : 안건우 박기춘 김성수

목 차

View – 담당 : 최인호

1. ChartPrimaryPanel

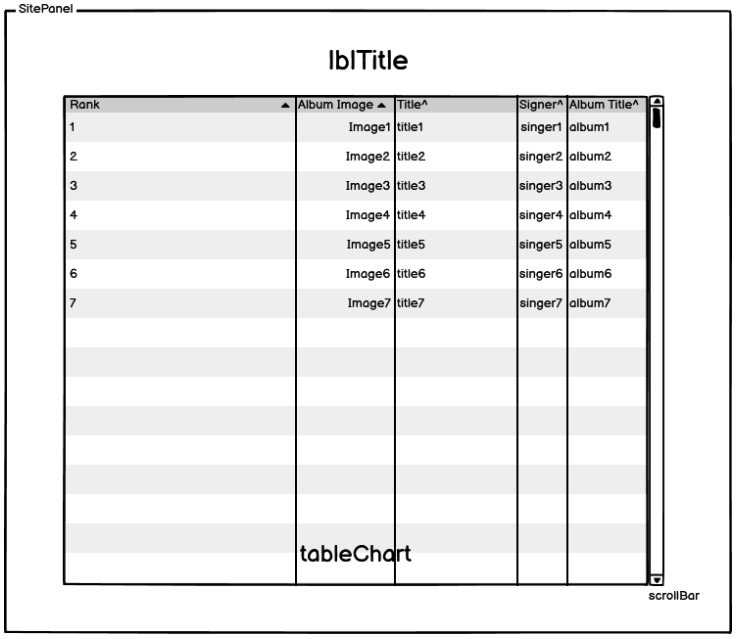


정보를 담고 있지 않은 JPanel을 상속받는 클래스

constructor에 너무 많은 멤버 변수가 초기화되어 있다.

각 멤버 변수별로 [위치 / 색 / 문장] 을 설정하는 메소드 분리 실시

2. SitePanel



ChartPrimaryPanel에서 객체를 생성해서 관리하고 있다.

AppManager을 통해서 정보를 받고 Chart에 뿌려주는 JPanel

constructor에 모든 멤버 변수가 초기화되어 있다.

쓸모 없는 멤버변수 줄이기

각 멤버 변수별로 초기화를 메소드로 분리

2-1. public void changeData()

```
public void changeData() {
   switch(AppManager.getS_instance().getSite_M_B_G()){
      case 1:
         strChartName = "Melon";
         break;
      case 2:
         strChartName = "Bugs";
         break;
      case 3:
         strChartName = "Genie";
         break;
   }
   lblTitle.setText(strChartName + " TOP 100");
   tableModel.setContents(AppManager.getS_instance().getParser().getChartList());
   buildTable();
   tableChart.repaint();
}
```

AppManager에서 getSite_M_B_G(1 멜론 | 2 벅스 | 3 지니)가 몇인지 받아서 switch문으로 각 사이트에 맞는 parser로 정보를 갖고 와서 chart에 보여준다.

switch문 제거

각 사이트 parser에서 데이터 갖고 오는 것을 메소드 분리

2-2. public void filter(String, Int)

ChartPrimaryPanel에서 txtSearch의 값에 따라서 chart를 필터 해주는 함수

무엇을 filter 해주는지 메소드 명 변경

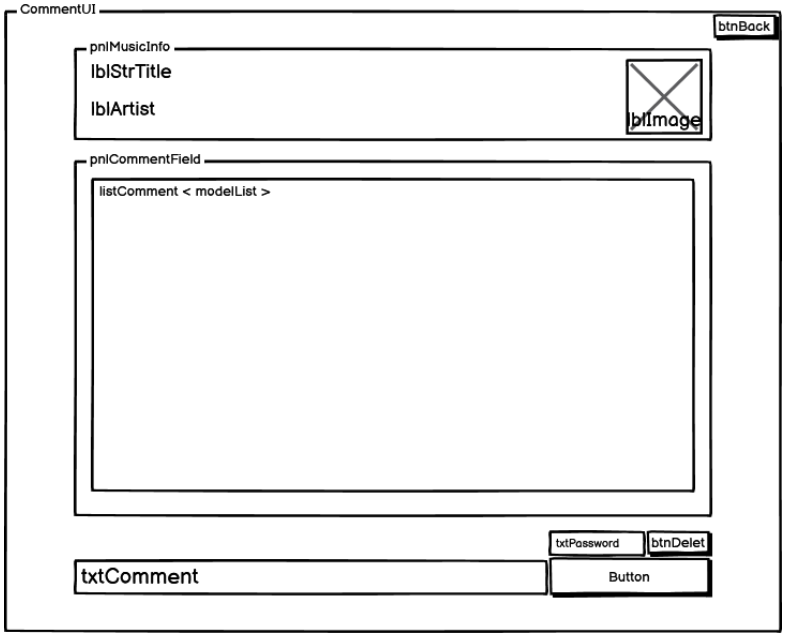
2-3. public class ChartModel()

tableChart의 데이터를 담고 있는 class

SitePanel 내에서만 쓰이기 때문에 private 선언 필요

3. CommentUI

~~Panel 형태로 클래스명 리팩토링 (쉽게 알 수 있도록)



chart에서 음원을 클릭하면 나오는 JPanel

AppManager에서 parser를 통해서 특정 음원정보를 Json으로 받아서 데이터를 처리한다.

AppManager에서 객체를 생성해서 관리하고 SitePanel에서 노래를 누르면 AppManager에서 화면을 전환하여서 보여주게끔 되어 있다.

Constructor에서 모든 멤버 변수를 초기화 하고 있다.

각 멤버변수 별로 메소드 분리

3-1. private void readComment()

DB에 접속해서 댓글과 password를 가져와서 modelList에 추가한다.

3-2. private void addMusicInfo(Int)

pnlMusicInfo안에 해당하는 멤버 변수들을 설정해주는 함수이다.

pnlMusicInfo를 클래스화 해서 분리

pnlMusicInfo내 멤버변수 설정하는 getter/setter 설정

AppManager를 통한 파라미터 제거

3-3. public void reNewalInfo(Int)

```
private void addMusicInfo(int rank) {
    String strRefinedTitle = strTitle;
    if (strRefinedTitle.indexOf("(") != -1) {
        strRefinedTitle = strRefinedTitle.substring(0, strRefinedTitle.indexOf("("));
    }
    lblTitle.setText(strRefinedTitle);

    String strRefinedArtist = strArtist;
    if (strRefinedArtist.indexOf("(") != -1) {
        strRefinedArtist = strRefinedArtist.substring(0, strRefinedArtist.indexOf("("));
    }
    lblArtist.setText(strRefinedArtist);
    Image image = null;
    URL url;

    try {
        url = new URL(AppManager.getS_instance().getDetailParser().getImageUrl());
        System.out.println(url);
        image = ImageIO.read(url);
        image = image.getScaledInstance(160, 160, Image.SCALE_SMOOTH);
    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    lblImage.setIcon(new ImageIcon(image));

}//addMusicInfo
```

AppManager에서 노래 선택이 바뀌어서 본 객체를 불러올 때 노래에 맞게끔 DB에 접속해서 본 객체의 모든 멤버 변수를 바꾼다.

> constructor에서 멤버변수 초기화하는 과정과 기능이 겹쳐 있다. 그러므로 중복코드를 제거하는 방향으로 리팩토링 예정

AppManager – 담당 : 최인호

1. 설명

Main함수에서 호출하며 MVC에 해당하는 모든 Class 객체를 생성 및 관리한다.

> Class 전체적으로 switch문 사용이 빈번하다. if문 혹은 메소드 분리로 switch문 제거

각 데이터를 전달해주고 View를 직접 관리한다.

> View에 해당하는 부분을 분리해야 한다. (MVC 패턴 명료화)
>
> class내 생성되는 PrimaryPanel을 클래스로 생성해서 만들어진 클래스에서 View를 관리하게끔 코드 및 메소드 분리

2. 핵심 멤버 변수

int Site_M_B_G : 지금 보여지고 있는 화면의 정보 (Melon = 1 | Bugs = 2 | Genie = 3)

final 선언으로 멜론 뮤직 벅스 이해하기 쉽게 리팩토링

JSONArray[] chartData[3] : 멜론 벅스 지니의 TOP100 차트 데이터를 갖고 있는 배열

변수명을 읽기 쉽게 바꾸는 리팩토링 필요

private MelonChartParser | private BugsChartParser | private GenieChartParser

각 음원 사이트의 TOP100 사이트를 parser 하는 멤버 변수

Data파트로 분리하는 리팩토링이 필요하다.

3. Constructor

parser를 생성하고 각 parser로부터 받을 JsonArray를 생성한다.

2-1.public void DataPassing(Component parentComponent)

AppManager외에는 사용하지 않는 메소드이므로 캡슐화 진행

2-2.public JSONArray getJSONArray(int index)

public JSONArray getJSONArray()

```
public JSONArray getJSONArray() {
    switch(Site_M_B_G) {
        case 0:
            return chartData[1];
        case 1:
            return chartData[2];
        case 2:
            return chartData[3];
        default:
            throw new IndexOutOfBoundsException("the length of chartData is 3");
    }
}
```

parser를 통해 얻은 데이터를 전달하는 방법이 다양하다.

일정 규칙을 정해서 하나로 통일

switch문 삭제

4. View 패턴 명료화

class내 생성되는 PrimaryPanel을 클래스로 생성해서 만들어진 클래스에서 View를 관리하게끔 코드 및 메소드 분리

Class 전체적으로 switch문 사용이 빈번하다. if문 혹은 메소드 분리로 switch문 제거

5. 일정

8주차 : 에러 및 Conflict 수정

9주차 : View 및 AppManager 리팩토링

10주차 ~ 13주차 : 앨범 별 Parser 생성 및 DB연결

JSOUP을 이용한 Parser생성

14주차 : 에러 수정 및 최종 점검

Database – 담당 : 박기춘

1. DB연결 스크립트

```
public static Connection GetDB() {
    Connection con = null;
    String url = "URL [ 길어서 생략 ]"
    String userid = "admin";
    String pwd = "refactoring";
    try { // 1. 드라이버 로딩
        Class.forName("com.mysql.jdbc.Driver");
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
    try { // 2. 연결
       con = DriverManager.getConnection(url, userid, pwd);
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return con;
}
```

다른 메서드에선 DB연결할 때 사용되는 메서드. DB에 연결된 connection을 반환한다.

2. 코멘트 UI 스크립트

```
public void reNewalInfo(int rank) {
    strTitle = AppManager.getS_instance().getParser().getTitle(rank);
    strArtist = AppManager.getS_instance().getParser().getArtistName(rank);

    JSONArray chartListData = AppManager.getS_instance().getParser().getChartList();
    AppManager.getS_instance().detailDataPassing(rank, chartListData, this);

    readComment();
    addList();
    addMusicInfo(rank);
}
```

AppManager에서 호출한 reNewalInfo(). 선택한 노래의 상세정보들을 파싱하여 곡정보를 저장하고 차례대로 함수들을 호출한다.

```
private void readComment() {
    con = ConnectDB.GetDB();
    sqltitle = strTitle;
    if (sqltitle.contains("'")) {
        sqltitle = sqltitle.replace("'", ":");
    }
    try {
        stmt = con.createStatement();
        String sql = "SELECT comment, pwd FROM songinfo WHERE title = '" + sqltitle + "'";
        rs = stmt.executeQuery(sql);
        while (rs.next()) {
            String comment = rs.getString("comment");
            String pwd = rs.getNString("pwd");
            arrComment.add(comment);
            arrPassword.add(pwd);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}//readComment
```

reNewalInfo가 호출한 readComment(). DB에 연결하여 해당 노래의 코멘트와 그 코멘트의 비밀번호를 불러온다. 불러온 각각의 정보들은 리스트에 저장한다. 노래제목에 작은 따옴표(')가 있는 경우 DB에서 오류를 발생시켜서 임시로 콜론(:)으로 대체하여 검색한다.

```
private void addList() {
    for (String ptr : arrComment) {
        modelList.addElement(ptr);
    }
    listComment.setModel(modelList);
    pnlCommentField.add(listComment);
}
```

reNewalInfo가 호출한 addList(). 리스트에 저장된 코멘트들을 차례대로 보여준다.

```
private void addMusicInfo(int rank) {
    lblTitle.setText(strRefinedTitle);
    lblArtist.setText(strRefinedArtist);
    try {
        url = new URL(AppManager.getS_instance().getDetailParser().getImageUrl());
        image = ImageIO.read(url);
        image = image.getScaledInstance(160, 160, Image.SCALE_SMOOTH);
    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    lblImage.setIcon(new ImageIcon(image));
}//addMusicInfo
```

reNewalInfo가 호출한 addMusicInfo(). 음악 상세 정보에 보여줄 데이터(노래제목, 가수, 앨범사진)들을 보여온다.

```
public void addBtnRegisterListener(ActionListener listenForBtnRegister) {
    btnRegister.addActionListener((listenForBtnRegister));
}
public void addBtnDeleteListener(ActionListener listenForBtnDelete) {
    btnDelete.addActionListener((listenForBtnDelete));
}
public void addBtnBackListener(ActionListener listenForBtnBack) {
    btnBack.addActionListener((listenForBtnBack));
}
```

각각의 버튼에 ActionListener 를 연결해주는 메서드.

3. 코멘트UI 컨트롤러 스크립트

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

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

댓글 등록을 하는 메서드. 댓글 작성 칸이 공백이 아니라면 DB에 노래의 상세정보들과 코멘트, 비밀번호를 등록한다. 비밀번호가 설정되어 있지 않다면 0000으로 자동 등록된다.

```
private class ButtonDeleteListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        if                    (Integer.parseInt(theCommentUI.txtPassword.getText())                    ==
Integer.parseInt(theCommentUI.arrPassword.get(theCommentUI.listComment.getSelectedIndex()))) {
         theCommentUI.modelList.removeElementAt(theCommentUI.listComment.getSelectedIndex());
            theCommentUI.con = ConnectDB.GetDB();
            try {
                String sql = "DELETE FROM songinfo WHERE title = ? AND pwd = ?";
                pstmt = con.prepareStatement(sql);
                pstmt.setString(1, theCommentUI.sqltitle);
                pstmt.setString(2, theCommentUI.txtPassword.getText());
                int temp = pstmt.executeUpdate();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
        theCommentUI.txtPassword.setText("");
    }//actionPerfomed
}//ButtonDeleteListener
```

댓글 삭제를 하는 메서드. 클릭한 코멘트의 저장된 비밀번호와 입력한 비밀번호가 일치하다면 저장된 리스트에서 해당 목록을 지우고 DB에 연결하여 해당 코멘트를 삭제한다.

```
private class ButtonBackListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        theCommentUI.clearAll();
        AppManager.getS_instance().BackToChartPrimaryPanel();
    }//actionPerfomed
}//ButtonBackListener
```

뒤로 가기 버튼의 메서드. UI화면을 지우고 AppManager에서 차트를 불러온다.

4. 리팩토링 계획 내용

4-1. 메서드 안에서 화면을 구성하는 코드들은 다른 메서드로 분리한다.

```
private void addList() {
    for (String ptr : arrComment) {
        modelList.addElement(ptr);
    }
    listComment.setModel(modelList);
    listComment.setFont(new Font("서울한강체 M", Font.PLAIN, 20));
    listComment.setBounds(0, 0, 960, 400);
    pnlCommentField.add(listComment);
}
```

addList() 메서드에 modelList 를 생성하는 코드와 패널에 코멘트를 보여주는 코드가 같이 있다.

4-2. 사용하지 않거나 기능이 없는 메서드, 변수들을 제거한다.

```
private class ButtonRegisterListener implements ActionListener {
        private Component _viewLoading;
        public ButtonRegisterListener() {
        }
        public ButtonRegisterListener(Component parentComponent) {
            _viewLoading = parentComponent;
        }

}//ButtonRegisterListener
```

Listener안에 사용되지 않는 코드가 존재한다.

4-3. 한 클래스에서 반복 사용되면서 새로 정의되는 임시변수들을 분리한다.

```
String sql = "INSERT INTO songinfo VALUES (?, ?, ?, ?, ?, ?)";
String sql = "DELETE FROM songinfo WHERE title = ? AND pwd = ?";
```

sql 쿼리문을 이용할 때마다 여러 리스너 안에서 String sql을 매번 새로 호출한다.

4-4. 한두번만 사용되는 임시변수들은 메서드 호출로 변경한다.

```
public void actionPerformed(ActionEvent e) {
    if (!theCommentUI.txtComment.getText().equals("")) {
        con = ConnectDB.GetDB();
        try {
            String sql = "INSERT INTO songinfo VALUES (?, ?, ?, ?, ?, ?)";
            pstmt = con.prepareStatement(sql);

//생략//
}//actionPerfomed
```

DB를 연결할 때마다 con = ConnectDB.GetDB()는 한번만 사용된다.

4-5. 반복되는 기능에 변수만 다른 기능을 메서드로 변경한다.

```
String strRefinedTitle = strTitle;
if (strRefinedTitle.indexOf("(") != -1) {
    strRefinedTitle = strRefinedTitle.substring(0, strRefinedTitle.indexOf("("));
}
lblTitle.setText(strRefinedTitle);

String strRefinedArtist = strArtist;
if (strRefinedArtist.indexOf("(") != -1) {
    strRefinedArtist = strRefinedArtist.substring(0, strRefinedArtist.indexOf("("));
}
lblArtist.setText(strRefinedArtist);
```

음악 정보를 보여주기 위해 리스트에서 곡제목과 가수의 String에 편집을 하는 부분이 동일하다.

5. 일정

9주차 : DB병합 오류 해결

10주차 : 코멘트 리펙토링-여러 기능을 하는 메서드를 추출하는 작업

11주차 : 코멘트 리펙토링-임시변수, 메서드 분리 및 제거

12주차 : DB 및 AppManager 리펙토링

13주차 : 추가된 기능들 리펙토링

14주차 : 최종정리

Model – 담당 : 안건우

1. 다이어그램

< 모델 클래스 다이어그램 >

< 파서 다이어그램 >

2. 코드 분석

MusicChartParser 를 Melon, Genie, BugsChartParser 가 상속받고 있다.

필드로는 songCount, chartList songDetailInfo, url, chartThread, songDetailThread 가있다.

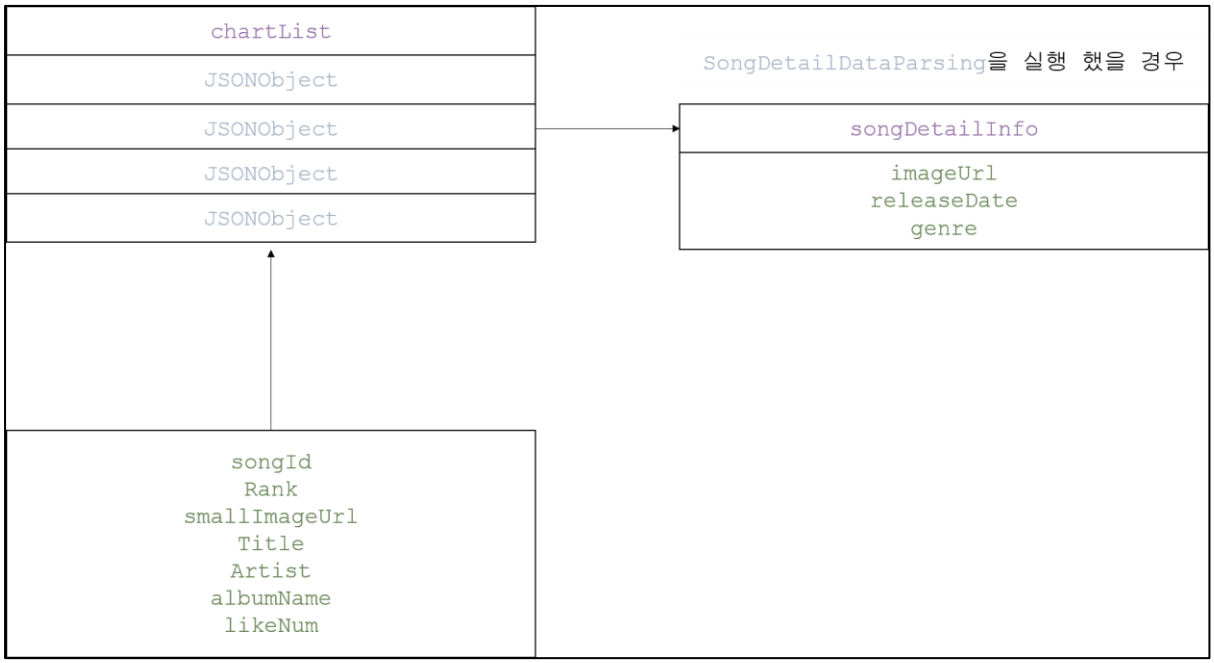
int songCount는 차트의 100곡을 파싱할 때 노래 개수를 카운트하는 변수이다.

String url 는 파싱 할 웹 사이트 url이다.

chartDataParsing, songDetailDataParsing메소드를 통해 chartList 로 데이터를 파싱할 수 있다. int getRank, String getTitle, String getArtistName, String getAlbumName, String getSongId 등으로 차트 데이터에 대한 접근을 할 수 있다.

파싱한 노래의 데이터는 랭크 순서대로 JSONArray chartList 에 담기며 노래 한 곡에 대한 상세 정보는 JSONObject songDetailInfo; 에 담겨있다. 멜론 차트를 예로 들어보면 chartList 의 JSONObject 하나 에는 songId, rank, smallImageUrl, title, artist, albumName, likeNum 이 담겨 있고 songDetailInfo 에는 imageUrl, releaseDate, genre 가 담겨있다. 각 chartList 의 JSONObject 와 songDetailInfo 에는 조금씩 다른 정보들이 들어있다.

예시로 MelonChartParser 의 데이터의 시각화는 다음과 같다.



< 데이터 모델 >

3. 리팩토링

멤버 변수의 값을 변경하는 메소드에는 chartDataParsing와 songDetailDataParsing가있다. setChartData, setSongDetailData 메소드로 메소드명을 변경시킨다.

중복되는 get메소드가 많다.

```
public String getAlbumName(int rank)
public String getAlbumName(String title)
public String getAlbumName(JSONObject jObj)
```

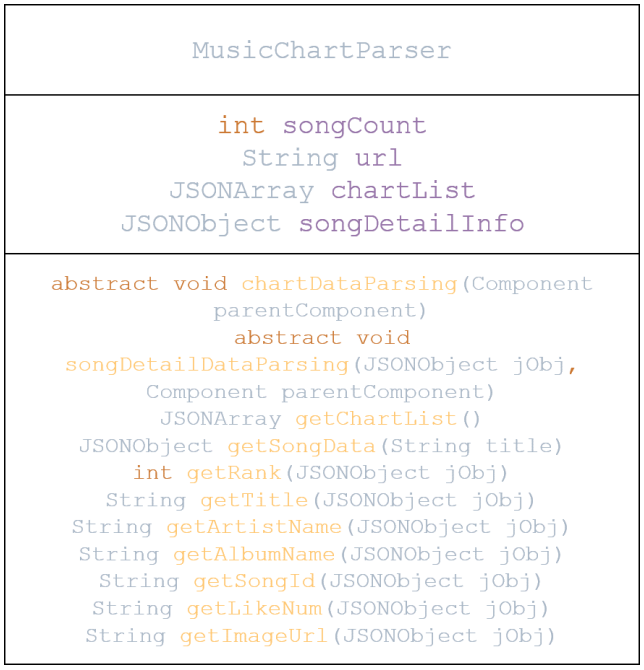
이 함수들은 코드내용이 비슷하고 JSONObject 안에는 rank,title 과 같은 데이터들이 들어 있으므로 알고리즘 전환을 통해 getAlbumName 메소드들을 getAlbumName(JSONObject jObj) 메소드로 합친다.

나머지 중복되는 get메소드들에 관해서도 같은 내용을 반복한다.

```
public abstract void chartDataParsing(Component parentComponent);
public abstract void songDetailDataParsing(String songId, Component parentComponent);
public abstract void songDetailDataParsing(JSONObject jObj, Component parentComponent);
public abstract void songDetailDataParsing(int rank, JSONArray chartListData, Component parentComponent);
public abstract void songDetailDataParsing(String title, JSONArray chartListData, Component parentComponent);
```

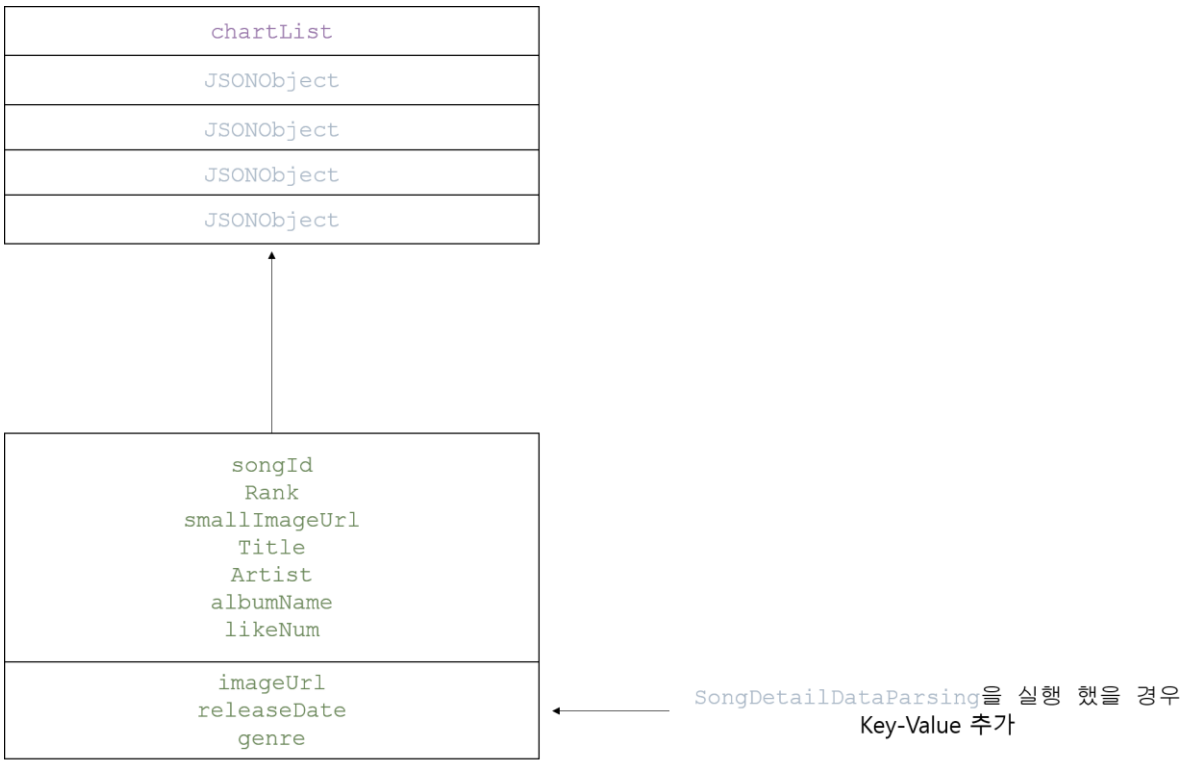
위 코드기능이 같고 JSONObject 에 rank,title 등이 들어있으므로 알고리즘 전환을 통해

songDetailDataParsing(JSONObject jObj, Component parentComponent); 만 남기고 나머지는 삭제한다. 이와 같이 중복되는 메소드 들을 삭제한후 다이어그램은 아래와 같다.



< 수정된 MusicChartParser 다이어그램 >

앞서 나왔던 MusicChartParser 의 데이터를 다음과 같은 형태로 수정한다.



< 수정된 데이터 모델 >

데이터 모델 수정 후 관련된 메소드들을 수정한다.

3. 일정

9 주차: 메소드명 변경

10주차: 알고리즘 전환을 통해 중복 메소드 병합

11주차~12주차: JSONArray ChartList 와 songDetailInfo 를 합치고 songDetailInfo 필드 삭제

songDetailInfo 필드 삭제로 인한 관련된 메소드 변경

13주차: 유튜브 스트리밍 기능 추가

유튜브 스트리밍 기능은 주 브라우저로 해당 노래의 유튜브 url 주소를 실행한다.

14주차: 유튜브 스트리밍 기능 리팩토링

리팩토링된 프로그램에 기능추가 과정 분석

Controller – 담당 : 김성수

1. ChartPrimaryPanelController 분석

해당 클래스는 ButtonRefreshListener, ButtonMelonListener, ButtonBugsListener, ButtonGenieListener, KeyActionListener의 구성으로 이루어져 있다.

1.1 ButtonRefreshListener

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

    @Override
    public void actionPerformed(ActionEvent e) {
        theChartPrimaryPanel.current = LocalDateTime.now();
        theChartPrimaryPanel.formatter = DateTimeFormatter.ofPattern("HH:mm:ss");
        switch (AppManager.getS_instance().getSite_M_B_G()){
            case 1:
                theChartPrimaryPanel.formatted_Melon =
theChartPrimaryPanel.current.format(theChartPrimaryPanel.formatter);
                theChartPrimaryPanel.lblTime.setText("Renewal time : " +
theChartPrimaryPanel.formatted_Melon);
                AppManager.getS_instance().setSite_M_B_G(1);
                AppManager.getS_instance().DataPassing(_viewLoading);
                System.out.println("why?");
                break;
            case 2:
                theChartPrimaryPanel.formatted_Bugs =
theChartPrimaryPanel.current.format(theChartPrimaryPanel.formatter);
                theChartPrimaryPanel.lblTime.setText("Renewal time : " +
theChartPrimaryPanel.formatted_Bugs);
                AppManager.getS_instance().setSite_M_B_G(2);
                AppManager.getS_instance().DataPassing(_viewLoading);
                break;
            case 3:
                theChartPrimaryPanel.formatted_Genie =
theChartPrimaryPanel.current.format(theChartPrimaryPanel.formatter);
                theChartPrimaryPanel.lblTime.setText("Renewal time : " +
theChartPrimaryPanel.formatted_Genie);
                AppManager.getS_instance().setSite_M_B_G(3);
                AppManager.getS_instance().DataPassing(_viewLoading);
                break;
        }
    }
}//ButtonRefreshListener
```

버튼 클릭 시 현재 Panel에 표시된 음원 사이트의 실시간 차트 정보를 다시 불러온 후 갱신한 시간을 Panel 오른쪽 하단에 표시해준다

1.2 ButtonMelonListener

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

   @Override
   public void actionPerformed(ActionEvent e) {

      if(AppManager.getS_instance().getSite_M_B_G() == 1) return;
      AppManager.getS_instance().setSite_M_B_G(1);
      if(!AppManager.getS_instance().getParser().isParsed())
AppManager.getS_instance().DataPassing(_viewLoading);
      System.out.println("Melon");
      theChartPrimaryPanel.pnlSitePanel.changeData();
      theChartPrimaryPanel.lblTime.setText("Renewal time : " +
theChartPrimaryPanel.formatted_Melon);
      theChartPrimaryPanel.txtSearch.setText("");
      theChartPrimaryPanel.pnlSitePanel.filter(null,2);
   }
}//ButtonMelonListener
```

버튼 클릭 시 불러왔던 멜론 사이트의 음원 차트와 갱신 시간을 panel에 표시해준다

1.3 ButtonBugsListener

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

    @Override
    public void actionPerformed(ActionEvent e) {

        if(AppManager.getS_instance().getSite_M_B_G() == 2) return;
        AppManager.getS_instance().setSite_M_B_G(2);
        if(!AppManager.getS_instance().getParser().isParsed())
AppManager.getS_instance().DataPassing(_viewLoading);
        System.out.println("Bugs");
        theChartPrimaryPanel.pnlSitePanel.changeData();
        theChartPrimaryPanel.lblTime.setText("Renewal time : " + theChartPrimaryPanel.formatted_Bugs);
        theChartPrimaryPanel.txtSearch.setText("");
        theChartPrimaryPanel.pnlSitePanel.filter(null,2);
    }
}//ButtonBugsListener
```

버튼 클릭 시 불러왔던 벅스 사이트의 음원 차트와 갱신 시간을 panel에 표시해준다.

1.4. ButtonGenieListener

```
private class ButtonGenieListener implements ActionListener {
    private Component _viewLoading;
    public ButtonGenieListener() { }
    public ButtonGenieListener(Component parentComponent){
        _viewLoading = parentComponent;
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        if(AppManager.getS_instance().getSite_M_B_G() == 3) return;
        AppManager.getS_instance().setSite_M_B_G(3);
        if(!AppManager.getS_instance().getParser().isParsed())
AppManager.getS_instance().DataPassing(_viewLoading);
        System.out.println("Genie");
        theChartPrimaryPanel.pnlSitePanel.changeData();
        theChartPrimaryPanel.lblTime.setText("Renewal time : " + theChartPrimaryPanel.formatted_Genie);
        theChartPrimaryPanel.txtSearch.setText("");
        theChartPrimaryPanel.pnlSitePanel.filter(null,2);
    }
}
```

버튼 클릭 시 불러왔던 지니 사이트의 음원 차트와 갱신 시간을 panel에 표시해준다

1.5 KeyActionListener

```
private class KeyActionListener implements KeyListener{
    private Component _viewLoading;
    public KeyActionListener() { }
    public KeyActionListener(Component parentComponent){
        _viewLoading = parentComponent;
    }
    @Override
    public void keyTyped(KeyEvent e) { }
    @Override
    public void keyPressed(KeyEvent e) { }
    @Override
    public void keyReleased(KeyEvent e) {
        Object obj = e.getSource();

        if(obj == theChartPrimaryPanel.txtSearch){
            //strSearchCategory = {"Name", "Artist"};
            if(0 == theChartPrimaryPanel.strCombo.getSelectedIndex())//Name
                theChartPrimaryPanel.pnlSitePanel.filter(theChartPrimaryPanel.txtSearch.getText(),2);
            if(1 == theChartPrimaryPanel.strCombo.getSelectedIndex())//Artist
                theChartPrimaryPanel.pnlSitePanel.filter(theChartPrimaryPanel.txtSearch.getText(),3);
        }//comboBox 0, 1일때 sitepanel의 filter에서 검색
    }//KeyReleased
}
```

검색창에 카테고리 설정에 따라 키보드 키를 입력할 때마다 해당 입력 값이 포함되는 곡들만 차트에 표시해준다

2 SitePanelController 분석

해당 클래스는 addClickListener가 포함되어 있다.

2.1 addClickListener

```
private class addClickListener implements MouseListener {
   private Component _viewLoading;
   public addClickListener() { }
   public addClickListener(Component parentComponent){
      _viewLoading = parentComponent;
   }

   @Override
   public void mouseEntered(MouseEvent e) { }
   @Override
   public void mouseExited(MouseEvent e) { }
   @Override
   public void mouseClicked(MouseEvent e) {
      Object obj = e.getSource();
      if(obj == theSitePanel.tableChart) {
         JTable table = (JTable) obj;
         Object[] music =
theSitePanel.tableModel.getMusicData(table.convertRowIndexToModel(table.getSelectedRow()))
;
         System.out.println(music[2] + music[0].toString());
         AppManager.getS_instance().PopUpCommentUI(Integer.parseInt(music[0].toString()));
      }
   }
   @Override
   public void mousePressed(MouseEvent e) { }
   @Override
   public void mouseReleased(MouseEvent e) { }
} //addClickListener
```

현재 차트에 표시된 곡을 클릭 시 CommentUI 패널을 불러오고 해당 곡의 이름과 가수 정보 및 해당 곡의 댓글 내용을 불러온다.

3. 리팩토링 계획

3.1 스위치문 정리

```
switch (AppManager.getS_instance().getSite_M_B_G()){
            case 1:
                theChartPrimaryPanel.formatted_Melon =
theChartPrimaryPanel.current.format(theChartPrimaryPanel.formatter);
                theChartPrimaryPanel.lblTime.setText("Renewal time : " +
theChartPrimaryPanel.formatted_Melon);
                AppManager.getS_instance().setSite_M_B_G(1);
                AppManager.getS_instance().DataPassing(_viewLoading);
                System.out.println("why?");
                break;
            case 2:
                theChartPrimaryPanel.formatted_Bugs =
theChartPrimaryPanel.current.format(theChartPrimaryPanel.formatter);
                theChartPrimaryPanel.lblTime.setText("Renewal time : " +
theChartPrimaryPanel.formatted_Bugs);
                AppManager.getS_instance().setSite_M_B_G(2);
                AppManager.getS_instance().DataPassing(_viewLoading);
                break;
            case 3:
                theChartPrimaryPanel.formatted_Genie =
theChartPrimaryPanel.current.format(theChartPrimaryPanel.formatter);
                theChartPrimaryPanel.lblTime.setText("Renewal time : " +
theChartPrimaryPanel.formatted_Genie);
                AppManager.getS_instance().setSite_M_B_G(3);
                AppManager.getS_instance().DataPassing(_viewLoading);
                break;
```

해당 스위치문에는 필요하지 않은 코드와 비슷한 채 반복되는 코드가 존재한다

3.2 같은 기능을 하는 메서드

`[ButtonMelonListener, ButtonBugsListener, ButtonGenieListener]` 세 메소드는 유사한 코드를 갖고 있어 하나의 메소드로 합칠 예정이다.

4. 일정

9주차 MVC 분리 오류 제거

10주차 스위치문 정리 (똑같이 사용되는 코드를 합치고 인자만 달라지는 부분을 정리함)

11주차 같은 기능을 하는 메서드 정리

12주차 최근 본 음악 목록 기능추가

13주차 추가 기능 리팩토링

14주차 최종점검