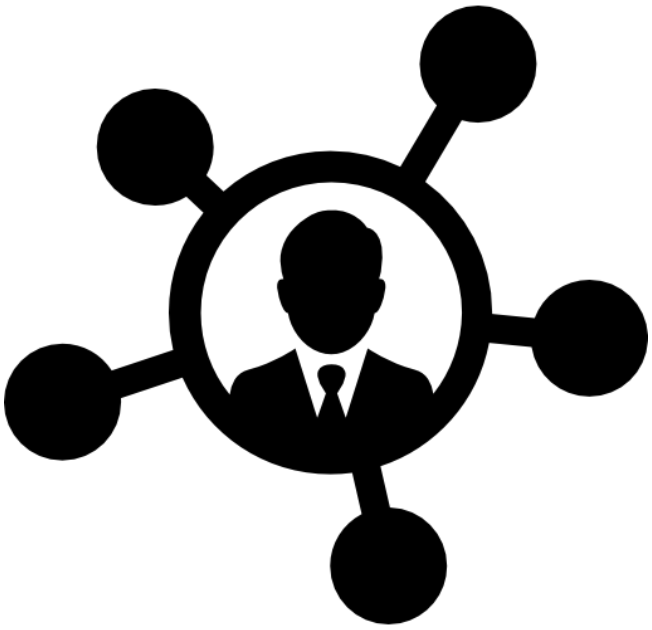
# 마인드맵 프로젝트



과목명 | 객체지향 프로그래밍
담당 교수 | 최지웅
학과 | 컴퓨터학부
학번 | 20172649, 20172646
이름 | 유정주, 양기조

# 목차

# [역할분담표]

| | |
|---|---|
| **Controller** | 유정주, 양기조 |
| Attribute_change_Listener | 유정주 |
| Attribute_Zone | 유정주 |
| Change_color_Listener | 유정주 |
| Dialog | 유정주 |
| **File_Chooser** | 유정주 |
| File_Chooser_Listener | 유정주 |
| MenuBar | 유정주 |
| **Mind_Tree** | 양기조 |
| Mindmap | 유정주, 양기조 |
| **Mindmap_Label** | 유정주 |
| **Mindmap_Zone** | 유정주, 양기조 |
| **MyPanel** | 양기조 |
| Other_Location_Click_Listener | 유정주 |
| Show_Tree_Click_Listener | 유정주 |
| Text_Zone | 유정주 |
| TextZone_apply_Listener | 유정주 |
| ToolBar | 유정주 |

*빨간 글씨는 중요 클래스, 이외는 기초 클래스

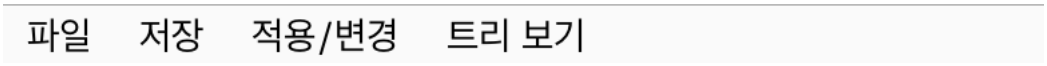
# [기초 클래스]

Mindmap
메인 함수가 들어있는 클래스이다. 이벤트 처리 클래스를 제외한 모든 프로그램을 구성하는  클래스, 레이아웃 클래스를 생성해준다. Controller를 설정해주고 각 클래스에 controller를 전달한다. 1280*720 크기의 프레임을 띄운다.

```
controller.set(TextmapZone, MindmapZone, AttributeZone, MindZone, MindTree, TextZone, this);
MindTree.setter_controller(controller);
TextmapZone.setter_controller(controller);
MindmapZone.setter_controller(controller);
AttributeZone.setter_controller(controller);
```

[그림 1] 컨트롤러에 TextZone, AttributeZone, MindZone, 두 개의 JSplitPane, 자기 자신의 정보를 입력해주는 코드

MenuBar




[그림 2] 메뉴바 그림(2)

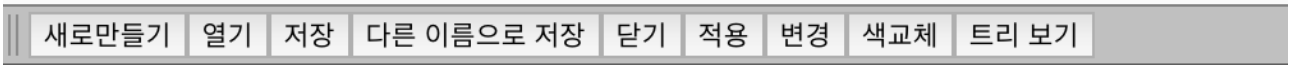

[그림 3] 메뉴바 그림(1)

메뉴바를 만든다. 메뉴바에 들어가는 아이템을 JMenuItem을 통해 만들고 JMenuBar에 넣는다. 모든 메뉴엔 ActionListener(Menu_Show_Tree_Listener,  File_Chooser_Listener, Attribute_change_Listener, Change_color_Listener, TextZone_apply_Listener)를 추가한다.

```
void action_item() {
    show_textarea.addActionListener(new Menu_Show_Tree_Listener());
    show_tree.addActionListener(new Menu_Show_Tree_Listener());
    save_save.addActionListener(new File_Chooser_Listener(controller));
    save_saveas.addActionListener(new File_Chooser_Listener(controller));
    file_open.addActionListener(new File_Chooser_Listener(controller));
    file_close.addActionListener(new File_Chooser_Listener(controller));
    file_new.addActionListener(new File_Chooser_Listener(controller));
    apply_change.addActionListener(new Attribute_change_Listener(controller));
    apply_change_color.addActionListener(new Change_color_Listener(controller));
    apply_apply.addActionListener(new TextZone_apply_Listener(controller));
}
```

[그림 4] ActionListener를 추가하는 코드

ToolBar




[그림 5] 툴바 그림

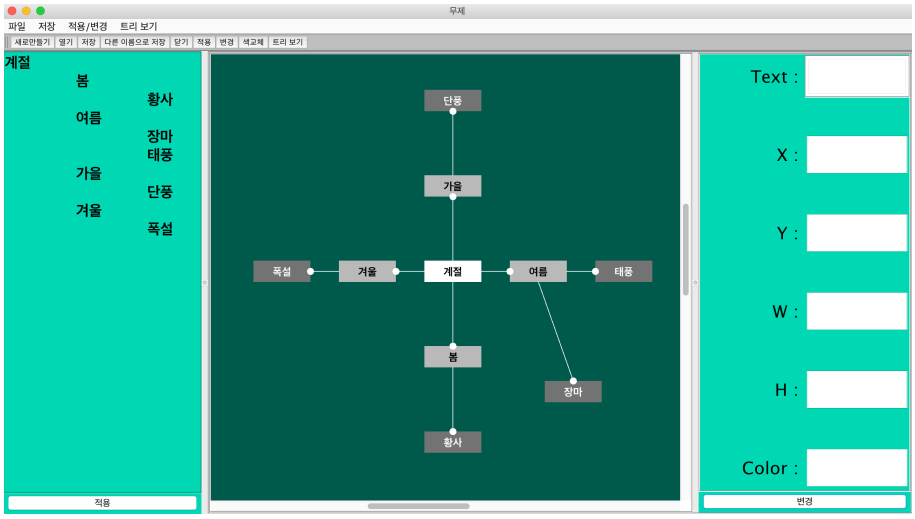
JToolBar를 생성한다. 여러 함수를 실행시키는 JButton을 만들고 ActionListener(File_Chooser_Listener, Attribute_change_Listener, Change_color_Listener, TextZone_apply_Listener, Show_Tree_Click_Listener)를 버튼에 추가한뒤 JToolBar에 추가한다.

Text_Zone
텍스트를 입력할 수 있는 JTextArea를 생성한다. JPanel을 만들고 그 위에 JTextArea를 가진 JScrollPane을 올린다.

TextZone_apply_Listener
Text_Zone의 적용 버튼에 사용되는 리스너이다. 입력한 텍스트가 존재하면 트리를 만들어 마인드맵을 출력하고 입력한 텍스트가 없어 StringIndexOutOfBoundsException이 던져지면 마인드맵을 모두 없앤다.

Attribute_Zone
클릭한 노드의 정보를 보여주는 구간. GridLayout으로 2행 6열의 칸을 만들고 좌측에는 JLabel을, 우측에는 JTextField를 배치한다. Text와 Color는 입력할 수 없다. X, Y, W, H는 키보드 입력을 통해 값을 변경할 수 있고 Color는 텍스트필드를 클릭하면 JColorChooser가 나오면서 색을 고를 수 있고 색을 선택하면 Color의 JTextField의 텍스트가 색상코드로 바뀐다. 값을 변경한 후 변경 버튼을 누르면 Attribute_change_Listener가 발생하고 노드의 정보가 변경된다.

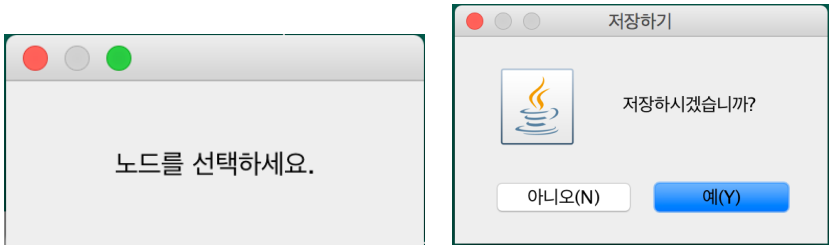
Attribute_change_Listener
attribute의 변경 버튼을 누를 때 발생하는 이벤트를 다루는 클래스. attribute의 x, y, w, h, color의 값을 Int와 Color로 받아 클릭한 노드의 정보를 변경한다.




[그림 6] 전체 화면 그림

Dialog

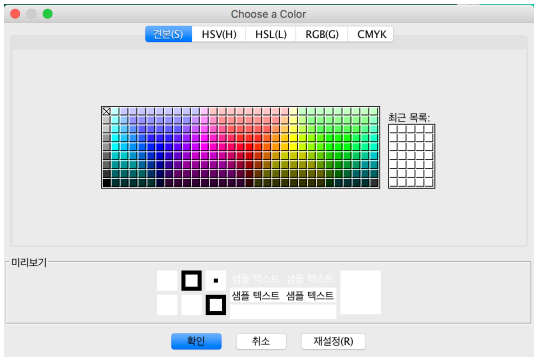
오류가 났을 때 다이얼로그를 띄워주는 클래스이다. 띄울 메시지를 인자로 넘겨 원하는 메시지가 나오는 다이얼로그를 만들어준다. 아무런 노드를 클릭하지 않고 Attribute Zone의 변경 버튼을 누르면 다이얼로그가 나온다.




[그림 7] 다이얼로그 그림

Change_color_Listener

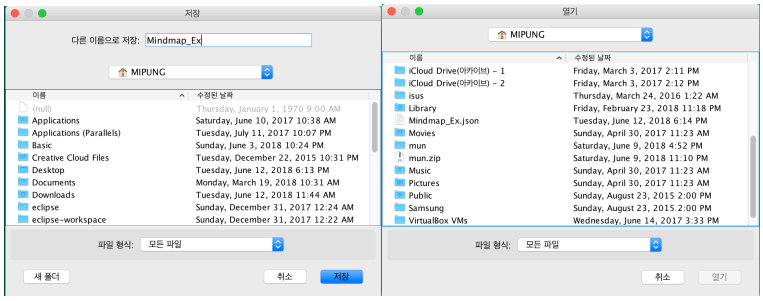
Mindmap Zone과 Text Zone, Attribute Zone의 색을 변경한다. Mindmap Zone의 색을 기준으로 brighter함수를 두 번 시행한 색을 Text Zone과 Attribute Zone 색을 변경한다. 컨트롤러를 통해 다른 클래스에 접근해 배경색을 바꾼다.




[그림 8] JColorChooser 그림

File_Chooser_Listener

누른 버튼이 무엇인지 판단하고 알맞은 함수를 실행할 수 있도록 도와준다. if else문을 사용하여 선택한 버튼이 뭔지 판별하여 알맞는 함수를 수행한다.




[그림 9] JFileChooser 그림

Other_Location_Click_Listener

노드의 선택 해제를 위해 만든 리스너이다. 현재 노드가 아닌 다른 곳을 선택하면 선택이 해제가 되고 8방향 JLabel이 show(false)가 된다. 다른 노드를 선택하면 전의 노드는 선택 해제가 되고 현재 누른 노드가 선택된다.

Show_Tree_Click_Listener
툴바와 메뉴바에 있는 트리보기-텍스트입력창보기 버튼의 리스너이다. 트리보기 상태일 때는 Text Zone에 JTree를 띄우고 텍스트입력창보기 상태일 때는 Text Zone에 JTextArea를 위치시킨다. 아무것도 입력하지 않았을 때 StringIndexOutOfBoundsException이 나와서 try-catch문으로 처리를 해주었다.

# [중요 클래스]

Controller
생성한 JSplitPane, ContentPane, Tree 등등 다른 클래스에 접근해야할
필요가 있는 클래스들의 주소를 저장한다. 마찬가지로 다른 클래스에도 controller를 저장한다. 이러한 Controller클래스를 통해 클래스간 접근이 가능하다.

File_Chooser

```
while (!BFS.isEmpty()) {
    q = (DefaultMutableTreeNode) BFS.poll();

    JSONObject node_Object = (JSONObject) json_arr.get(j++);
    Long x_var = (Long) node_Object.get("x_var");
    Long y_var = (Long) node_Object.get("y_var");
    Long w_var = (Long) node_Object.get("w_var");
    Long h_var = (Long) node_Object.get("h_var");
    Long color_var = (Long) node_Object.get("color_var");

    ((Mindmap_Label) (q.getUserObject())).setter_node(x_var.intValue(), y_var.intValue(),
            w_var.intValue(), h_var.intValue(), new Color(color_var.intValue()));

    for (int i = 0; i < q.getChildCount(); i++) {
        BFS.offer((DefaultMutableTreeNode) q.getChildAt(i));
    }
}
```

[그림 10] BFS로 노드를 순회하며 노드에 정보를 적용시킨다.

JSON을 이용하여 노드의 정보를 파일로 저장한다. 생성자에서는 파일만 선택할 수 있도록 설정해주고 다중선택을 불가능하게 하고 확장자가 json인 파일만 가져올 수 있도록 설정했다. open_file 함수에서는 BFS로 노드를 순회하면서 노드에 정보를 적용시킨다. save_file함수에서는 저장하시겠습니까?라는 YES_OR_NO 다이얼로그를 띄우고 저장을 누르면 BFS로 노드를 순회하며 json 파일에 저장하는 make_info를 실행한다. saveas_file은 다른 이름으로 파일을 저장할 수 있게 해주고 save_file과 작동 방식은 동일하다.

Mind_Tree

JTree를 저장한다. make_tree 메소드에서 텍스트필드에 올라와있는 내용을 while문을 통해 '\0'을 만날 때 까지 charAt()을 이용하여 읽는다. 탭을 읽으면 Level이 증가하고 엔터를 읽으면 다른 노드를 만든다.  이 노드를 이용해 만든 DefaultMutableTreeNode로 트리를 만든다. 트리를 만들면 controller를 통해 Mindmap_Zone이나 tree를 사용하는 다른 클래스에 전달한다. 만든 트리를 기반으로 JTree를 구축한다. JTree를 클릭하면 Mindmap에서 노드가 클릭되면서 8방위 레이블이 생긴다.


[그림 11] 구축된 JTree

Mindmap_Label

노드를 만드는 클래스이다. node JLabel과 8방향을 가리키는 8개의 JLabel로 구성되어 있다. 8개의 JLabel은 노드를 둘러싼 검은 테두리처럼 보이게 설정했고 MouseDragged가 불려져 노드의 크기를 변경시킨다. 각 라벨에는 사용자를 위해 툴팁을 넣어줬다. 노드가 Pressed가 되면 색이 반전되고 노드가 Clicked되면 8방향 JLabel이 보여진다.  색이 반전되는 것은 Color의 특징을 이용하여 각 RGB값을 최대값인 255에서 빼줘 설정해줬다.



[그림 12] 테두리 모양으로 이루어진 8개의 JLabel / [그림 13] Press된 노드

Mindmap_Zone

노드를 올리는 JPanel을 제공하는 클래스. 스크롤을 활성화시키기 위해 약 3배의 크기를 기본값으로 설정하고 (1000,1000)으로 이동시켜 JPanel을 중앙으로 이동시킨다. 오버라이딩한 paint를 사용하기 위해 My_Panel 객체를 만든다. 디자인을 위해 화살표는 동그란 연결점으로 대체하였다.

make_tree 메소드에서 매개변수로 트리의 root가 되는 DefaultMutableTreeNode를 받아와 패널에 노드를 올린다. 패널에 노드를 배치하는 로직은 트리의 root, 즉 마인드맵의 주제가 되는 노드가 하나임을 가정한다. root가 되는 노드를 패널의 중앙에 배치하고 root의 자식이 되는 노드들을 BFS로 순회하며 노드를 배치한다. 노드들은 트리의 깊이에 따라 색을 다르게 하여 배치한다. BFS로 순회한 노드가 이전에 순회한 노드와 깊이가 다르다면, 즉 깊이가 달라진다면 깊이가 하나 낮은 노드들의 위치를 구해 노

드 사이사이의 각도를 구하여 배치해야하는 노드의 부모와 부모의 이전 형제 사이부터 배치해야하는 노드의 부모와 부모 다음 형제 사이에 노드를 배치하는 방식이다. 노드들은 동심원의 형태로 배치되며 배치할 노드의 깊이가 달라질 때 반지름을 키우고 각도를 구해 Math의 sin, cos메소드를 이용하여 배치할 위치를 구해 노드를 배치한다.

MyPanel
JPanel을 상속하는 그림을 그리기 위한 패널을 제공하는 클래스 paint메소드를 오버라이딩한다. 오버라이딩한 paint메소드는 BFS로 노드들을 순회하며 배치된 노드의 부모의 위치를 구하고 배치된 노드의 위치를 구해 drawLine메소드로 선을 긋는다. 노드간 연결되는 선은 노드의 모서리에서 모서리로 이어지게 한다. 만약 트리가 존재하지 않으면 선을 그리지 않는다.

# [개발 일지]

5월 8~10일
9, 14단원 책 공부

5월 10일
레이아웃을 함수로 이용해서 완성하였다. 필요한 클래스와 관계, 어떤 클래스가 어떤 클래스를 상속할지, 수행 메뉴얼 등에 대해 회의하였다.

5월 16일
레이아웃을 클래스로 나눠서 구성했다. 내가 원하는 컴퍼넌트를 반환하게 클래스를 짜는 방법을 생각하지 못해 며칠을 해맸는데 16일에 드디어 성공했다. 툴바의 변경 버튼을 눌러 ColorChooser가 나오게 만들었다. MindMapZone의 색을 바꾸는 용도로 이용할 계획이다.

5월 17일
다른 클래스의 멤버의 색을 바꾸는 방법을 알 수 없어 MindMapZone을 더블클릭하면 색을 바꿀 수 있도록 바꾸었다.

5월 18일
원할한 이벤트 처리를 위해 툴바, 메뉴바, 텍스트존 등 구간 단위로 나눈 클래스를 레이아웃 세팅, 값 대입 등 동작 위주로 클래스를 나눠 코드를 재작성했다.

5월 20일
Attribute_Zone의 Color 텍스트필드를 클릭하면 색상팔레트가 나오고 색을 고르면 색상코드가 텍스트필드에 적히는 기능을 구현했다. 아마도 다시 클래스를 다른 방법으로 나눠야할 거 같다.

5월 23일
controller를 이용하여 다른 클래스의 접근을 성공했다. 처음으로 MVC 형식에 접근하였다. 다른 클래스에 접긴하기 위해 MVC를 도입해 controller를 고안하고 controller를 만들었다. 하지만 정확한 MVC 모델로 만든 것은 아닌 것 같다.

5월 24일
controller에 매개변수를 전하는 방법으로 수정했다. 텍스트존의 적용버튼을 누르면 마인드맵존에 라벨이 나오고 그 라벨을 누르면 x,y,h,w,color 값이 출력된다. 하지만 값을 변경하는 것은 실패했다. 다음 시도에 완성해야 할 것이다. 기존에 다른 기능들을 테스트하기위해 넣어두었던 노드를 직접삽입하는 부분을 대신해서 트리를 만들기 시작했다. 텍스트에서 읽어 만들은 노드들을 메인판에 트리의 형태로 전달하기 위해 TreeNode 인터페이스를 implements를 사용하는 LabelTreeNode class를 만들었다. (트리의 기능을 구현) 텍스트에서 노드들을 읽어 트리를 만드는 알고리즘을 구현했다.

5월 25일
레이블의 텍스트가 길어질 경우 Attribute의 TEXT에 다 들어가지 않으므로 TEXT를 스크롤로 만들었다. 레이블을 새로 만들 때 모든 레이블의 위치가 초기화된다. 이유는 revalidate로 인한 것 같은데 해결하지 못했다.

5월 26일

레이블의 위치가 초기화되는 것은 mindmap의 레이아웃을 null로 설정하여 해결하였다. 오늘은 레이블의 크기를 조정할 수 있는 4개의 버튼을 만들고 실현시켰다. 버튼은 노드레이블을 더블클릭하면 보이고 다시 한 번 클릭하면 보이지 않게 만들었고 크기를 3씩 증가시킨다.

5월 28일
1차로 JTree를 이용해 TextArea의 내용을 불러들여 마인드맵에 레벨에 맞춰 표시하도록 만들었다. 하지만 마인드맵처럼 배치를 하지는 못했다. 추후 수정을 해야하는 부분이다. JTree를 사용하기 위해 LabelTreeNode를 DefaultMutableTreeNode로 대체하였다. 노드의 깊이에 따라 색을 다르게 배치하도록 하였다. 아직까진 노드들이 메인판 가운데 몰려서 생성되었다.

5월 29일
2차 JTree 제작. 텍스트존의 적용버튼을 계속 누르면 중복되서 레이블이 나왔는데 removeall 함수를 이용하여 mindmap 위의 모든 레이블을 없애는 것으로 해결했다. JTree를 LabelTreeNode라는 커스텀클래스를 생성했는데 DefaultMutableTreeNode로 재생성했다. JTree를 직접적으로 이용해 마인드맵을 구성하는 방법은 찾지못해 간접적으로 이용하기로 했다. 툴바에 버튼을 하나 더 만든 뒤 그 버튼을 누르면 TextArea가 트리로 바뀌고 다시 한 번 누르면 TextArea로 바뀌게 하였다.

5월 30일
레이블 클릭으로 노드의 크기를 줄였다 늘였다한 액션을 드래그로 구현하였다.
트리의 위치를 적절히 배치하기 위한 알고리즘을 짰다. BFS를 이용해 트리를 읽고 배치하도록 만들었다. (BFS구현) 원리는 다음과 같다. 노드들을 원의 형태로 삽입한다. 노드의 수가 많아지면 원의 반지름을 늘리고 노드 사이의 간격을 줄인다. 다음 레벨의 노드들은 자신의 부모 위치로부터 다시 원의 형태로 노드를 삽입한다. 앞으로 노드들이 부모와 부모의 형제들이 그리는 원 내로 들어가지 않도록 부모의 양옆 형제로부터 일정 반경 사이에 노드들이 생기지 않도록 조치할 예정이다.

5월 31일
노드가 부모의 형제로부터 떨어져 생기도록 구현하였으나 치명적인 결함을 찾았다.
노드가 부모의 형제와 떨어져 부모의 부모에 가까이에 노드가 생겼고 자칫잘못하면 무한루프를 돌 수 있는 코드가 되었기 때문이다.

6월 1일
기존의 코드를 갈아엎고 새로운 방식의 노드배치를 사용했다. 메인판의 가운데로부터 노드를 생성한다. 기존의 방식대로 원의 형태로 노드를 생성하나, 레벨이 깊어질때마다 반경이 큰 형태의 원모양으로 노드들을 삽입한다. 다만, 부모가 자식을 많이 가진경우 해당 부모노드를 부모로 가지는 자식들이 좀 더 붙어서 생성되도록 한다.

6월 4일
선 그리기 기존에 쓰던 JPanel을 대체하고 그림을 그릴 수 있는 MyPanel 클래스를 만들어 mindmap_pane의 JPanel을 대체하였다.

6월 5일
파일 저장, 다른 이름으로 저장, 선 그리기를 구현했다. BFS를 이용해 모든 노드를 순회하며 노드의 정보를 저장하도록 하였다. 노드의 중앙에서 중앙으로 이어지던 기존의 선 그리기를 노드의 모서리에서 모서리로 이어지게 고침

6월 6일
선그리기 버그를 수정했다. 파일 오픈, 새로만들기를 구현했다. 기존 노드 배치가 조금씩 밀려생기거나 엉뚱한 위치에 생기는 버그를 고치기 시작하였고, 특정사항들에 대한 예외사항을 만들기 시작하였다

6월 7일
노드 배치를 완성하였다.

6월 8일
잔버그 해결하는 것으로 개발을 종료한다.