■ 과 목 명 : 객체지향언어2(8)

■ 담 당 교 수 : 유상미

■ 제 출 일 : 2024.12.21

■ 학 과 : 모바일소프트웨어트랙

■ 학 번 : 2191276

■ 성 명 : 이재혁

■ 과 제 번 호 : 최종 프로젝트 개발 보고서

자바를 이용한 자료구조 GUI

2191276 이재혁

1. 작품 개요

이 프로젝트는 자바를 이용한 자료구조 시뮬레이션 시스템으로, 사용자가 GUI를 통해 스택(Stack) 또는 큐(Queue) 자료구조를 선택하여 삽입과 삭제 과정을 시각적으로 확인할 수 있도록 설계되었다.

메인 화면에서 사용자는 버튼을 통해 두 자료구조 중 하나를 선택할 수 있으며, 선택한 자료구조에 대해 삽입 및 삭제 작업을 수행하면 그 과정이 화면에 직관적으로 표시된다.
스택의 경우, 데이터가 상단에 추가되고 제거되는 모습을 보여주며, 큐의 경우, 데이터가 들어오고 나가는 순서를 표현한다.

이 프로그램은 자료구조를 처음 배우는 학생, 초보 프로그래머, 혹은 자료구조의 개념을 시각적으로 가르치고자 하는 교육자들에게 유용하다.
GUI는 Java Swing으로 구현되었으며, 스택과 큐의 동작은 각각의 클래스로 나누어 모듈화하여 유지보수와 확장성을 고려했다.

추가적으로, 스택의 상태를 파일에 저장하거나 불러오는 기능을 통해 프로그램의 활용도를 높였으며, 향후에는 트리(Tree)나 그래프(Graph)와 같은 자료구조를 추가하여 더 폭넓은 학습 도구로 발전시킬 수 있는 확장 가능성을 제공한다.

2. 요구 및 기능 분석

요구 분석

- 교육적 요구: 자료구조의 동작 과정을 시각화하여 학습 효과를 극대화해야 한다.
- 사용자 요구: 사용자가 간단히 조작할 수 있는 직관적인 인터페이스가 필요하다.
- 확장 가능성: 향후 다른 자료구조를 추가할 수 있는 구조적 설계가 요구된다.

기능 분석

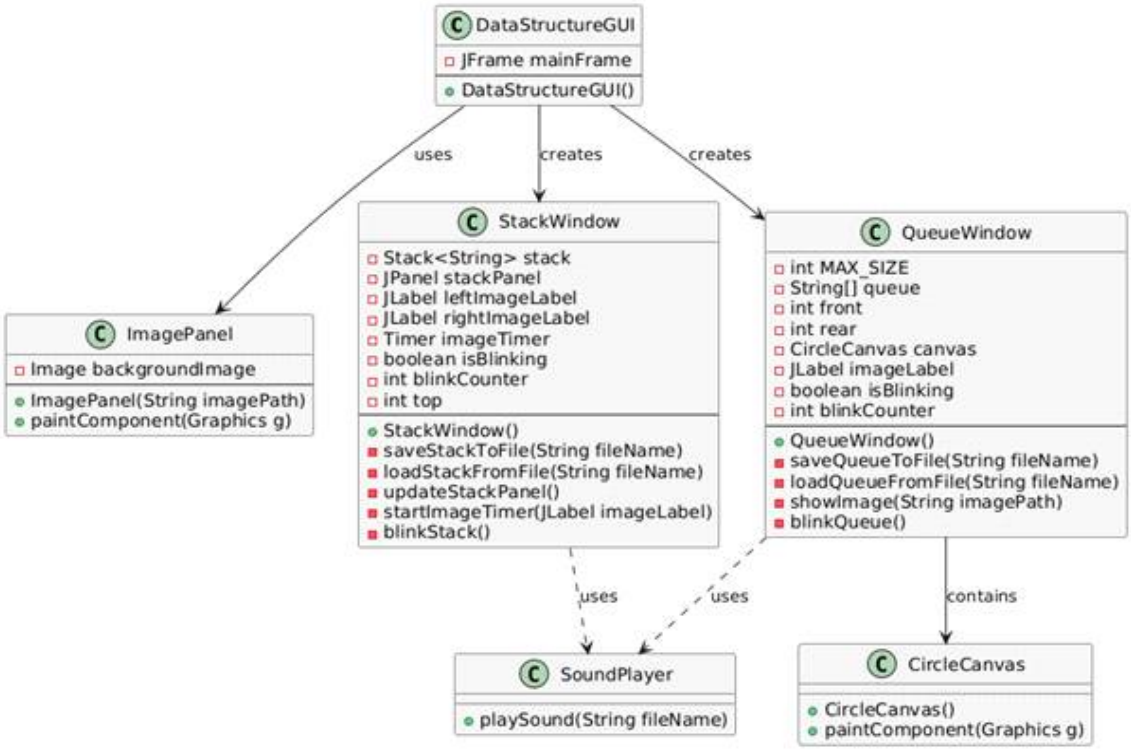
- 스택과 큐의 시뮬레이션: 삽입 및 삭제 연산이 시각적으로 표현한다. 또한 자료구조의 상태를 파일로 저장하고 불러올 수 있다.

- GUI 구현: Java Swing을 이용하여 사용자가 쉽게 조작할 수 있는 인터페이스를 제공한다. 또한 애니메이션 효과를 통해 데이터의 추가 및 제거가 강조된다.

- 오류 처리: 자료구조가 비어있거나 가득 찼을 때 사용자에게 메시지와 소리 알림을 제공한다

3. 설계

3.1 시스템



DataStructureGUI Class: 메인 GUI를 담당하며, 스택 및 큐 시뮬레이션 창으로 이동할 수 있는 인터페이스를 제공한다.

StackWindow Class: 스택의 삽입과 삭제 동작을 구현하고 시각적으로 표현한다.

QueueWindow Class: 큐의 삽입과 삭제 동작을 구현하고 시각적으로 표현한다.

CircleCanvas Class: 큐(queue)의 상태를 원형으로 시각화하는 역할을 한다.

ImagePanel Class: 이미지 배경을 지원하는 역할을 한다.

SoundPlayer Class: 사운드 파일을 재생하는 역할을 한다.

3.2 GUI 설계

메인 화면: "Stack" 및 "Queue" 버튼이 배치되어 사용자가 원하는 자료구조를 선택할 수 있다.

3.3 Stack 설계

화면: 상단에 메시지와 하단에 “삽입”, “삭제”, “저장”, “불러오기” 버튼이 배치되어 사용자가 직접 Stack을 구현할 수 있다. 이는 JButton을 사용해서 표시한다.이는 JButton을 사용해서 표시하고 JPanel에 추가하여 BorderLayout 아래에 배치한다.

Stack 구현: 최대 사이즈가 10인 스택을 구현할 수 있다. 아래에서 점점 위로 쌓는 방식입니

다. 새로운 데이터가 추가될 때 해당 데이터는 반짝거린다. 데이터를 더 이상 추가하거나 삭제를 하지 못 할 경우 경고음이 발생한다. JLabel을 이용해 데이터가 차곡차곡 쌓이는 모습을 시각적으로 표현한다. 각 데이터는 상단에 추가되며, 삭제 시 상단부터 제거된다. 또한 Java에서 제공하는 스택 구현 클래스를 이용해 데이터를 저장한다.

이미지: 화면 왼쪽에 삽입됐다는 이미지가 2초 동안 나타난다. 데이터가 삭제될 때에는화면 오른쪽에 삭제되었다는 이미지가 2초 동안 나타난다. 이는 JLabel과 Timer를 이용한다
저장 및 불러오기: 사용자가 저장 버튼을 누르면 스택의 현재 상황을 txt파일에 저장한다. 불러오기 버튼을 누르면 기존에 있던 스택을 비우고 txt파일에 저장된 데이터를 불러온다. 저장은 현재 스택의 상태를 텍스트 형식으로 저장하며, 각 데이터는 줄바꿈을 사용해 구분된다. 불러오기 시 저장된 데이터를 순서대로 읽어들여 자료구조를 재구성한다. 저장 기능은 BufferedWriter를 사용해 데이터를 파일에 기록하며, 불러오기 시에는 BufferedReader로 파일을 읽어들여 자료구조에 삽입한다.

3.4 Queue 설계

화면: 상단에 메시지와 하단에 “삽입”, “삭제”, “저장”, “불러오기” 버튼이 배치되어 사용자가 직접 Queue를 구현할 수 있다. 이는 JButton을 사용해서 표시하고 JPanel에 추가하여 BorderLayout 아래에 배치한다.

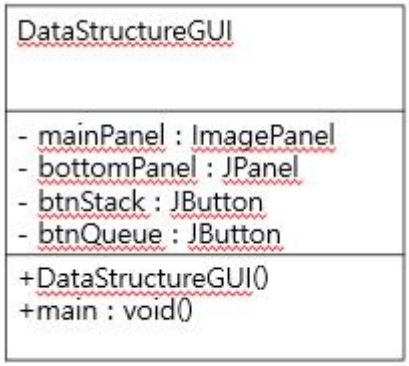
Queue 구현: 최대 사이즈가 10인 큐를 구현할 수 있다. 새로운 데이터가 추가될 때 해당 데이터는 반짝거린다. 데이터를 더 이상 추가하거나 삭제를 하지 못 할 경우 경고음이 발생한다. 큐는 JPanel에 그래픽으로 표현했고 이미지가 보일 수 있게 JLayeredPane을 사용해 레이어드를 추가했다. 그리고 JLayeredPane을 BorderLayout CENTER에 배치했다

이미지: 화면 가운데에 데이터가 정상적으로 삽입됐다는 이미지가 1초 동안 나온다. 데이터가 삭제될 때에는 화면 가운데에 삭제되었다는 이미지가 1초 동안 나온다.

저장 및 불러오기: 사용자가 저장 버튼을 누르면 큐의 현재 상황을 txt파일에 저장한다. 불러오기 버튼을 누르면 기존에 있던 큐를 비우고 txt파일에 저장된 데이터를 불러온다. 저장은 현재 큐의 상태를 텍스트 형식으로 저장한다. 각 데이터는 줄바꿈을 사용해 구분된다. 또한 front와 rear 값도 같이 저장한다. 불러오기 시 저장된 데이터를 순서대로 읽어들여 자료구조를 재구성한다. 저장 기능은 BufferedWriter를 사용해 데이터를 파일에 기록하며, 불러오기 시에는 BufferedReader로 파일을 읽어들여 자료구조에 삽입한다.

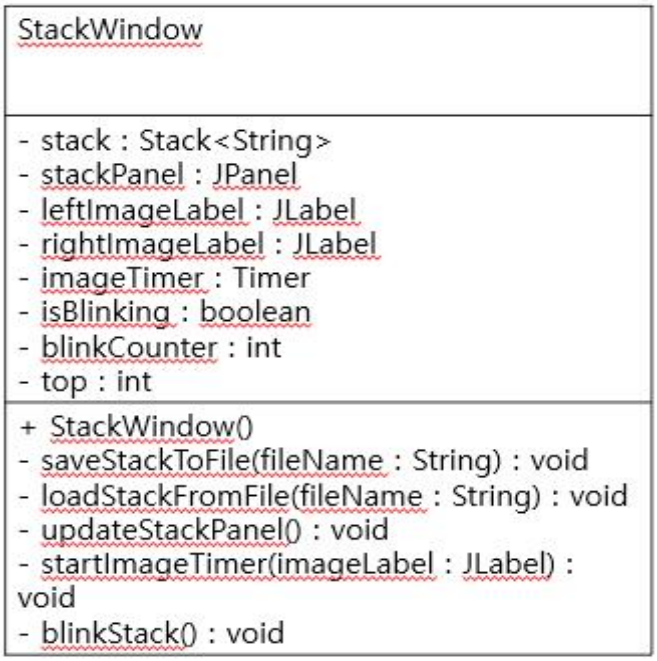
4. 프로그램 구현

4.1 DataStructureGUI



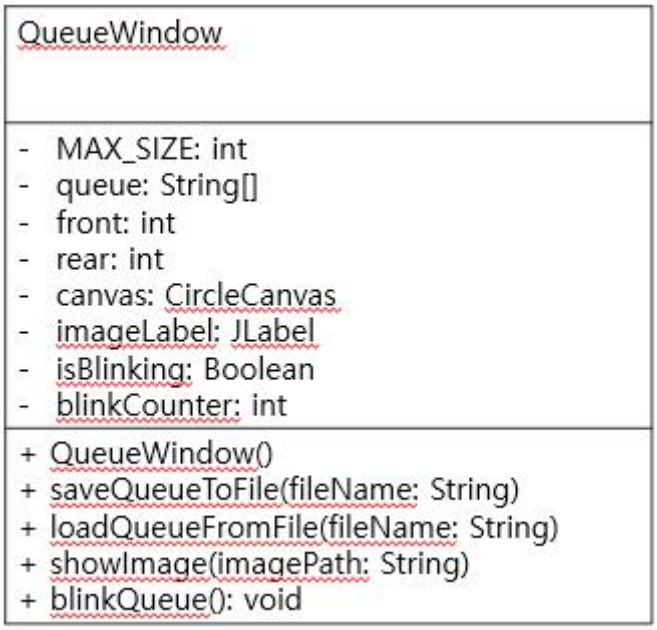
ImagePanel mainPanel: [배경 이미지를 표시하는 커스텀 패널이다.]
JPanel bottomPanel: [버튼이 위치한 하단 패널이다.]
JButton btnStack: [누르면 스택 윈도우가 열리는 스택 버튼이다.]
JButton btnQueue: [누르면 큐 윈도우가 열리는 큐 버튼이다.]

4.2 StackWindow



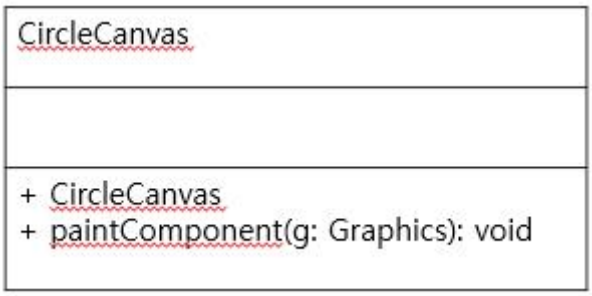
Stack<String> stack: [스택 데이터 구조]
JPanel stackPanel: [스택 데이터를 시각화하는 패널]
JLabel leftImageLabel: [삽입 시 이미지]
JLabel rightImageLabel: [삭제 시 이미지]
Timer imageTimer: [이미지에 타이머를 지정한다]
boolean isBlinking: [스택의 깜빡임 상태를 알려준다]
int blinkCounter: [스택의 깜빡임을 카운터하는 변수이다]
int top: [스택의 상단 인덱스 초기화한다]

4.3 QueueWindow

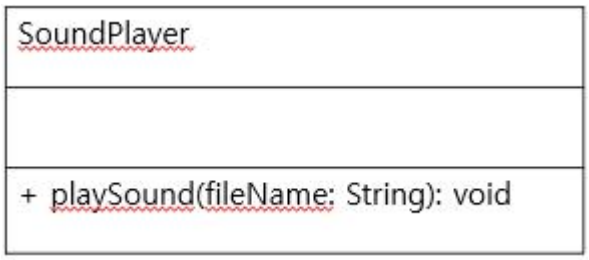


int MAX_SIZE: [큐의 최대 크기를 지정한다]
String[] queue: [큐의 데이터를 저장할 배열이다]
int front: [큐의 첫 번째 원소 인덱스이다]
int rear: [큐의 마지막 원소 인덱스이다]
CircleCanvas canvas: [큐를 시각적으로 표현할 캔버스이다]
JLabel imageLabel: [삽입/삭제 이미지 표시용 레이블이다]
boolean isBlinking: [깜빡임 여부를 알려준다]
int blinkCounter: [큐의 깜빡임을 세준다]

4.4 CircleCanvas



4.5 SoundPlayer



4.6 ImagePanel

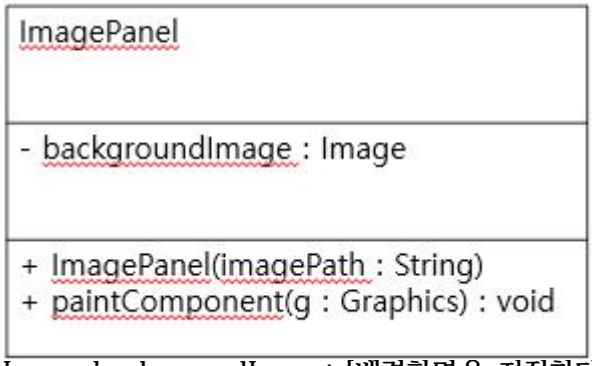


Image backgroundImage: [배경화면을 지정한다]

5. 주요 기능 설명

5.1 주요 구현 요소

스택 시뮬레이션:

push 메서드를 통해 데이터를 스택에 추가하고, pop 메서드를 통해 최상단 데이터를 제거한다. 이를 GUI 패널에 반영한다. 아래에서 위로 쌓기 위해 임의의 공간을 채워주었다. JLabel을 이용해 스택의 요소들을 표시했다.

<스택을 시각적으로 표현하는 메서드>

```
private void updateStackPanel() {
            stackPanel.removeAll();
            stackPanel.setLayout(new BoxLayout(stackPanel, BoxLayout.Y_AXIS));

            // 내용을 하단 정렬한다
            stackPanel.add(Box.createVerticalGlue());

            // 스택의 요소 표시한다
            for (int i = stack.size() - 1; i >= 0; i--) {
                JPanel itemPanel = new JPanel();
                itemPanel.setLayout(new BoxLayout(itemPanel,
BoxLayout.X_AXIS));
                itemPanel.setAlignmentX(Component.LEFT_ALIGNMENT);
```

```
                // TOP 인덱스에 해당하는 경우 "TOP" 라벨을 추가한다
                JLabel topLabel = new JLabel(" ");
                if (i == top) {
                    topLabel = new JLabel("TOP: " + i);
                    topLabel.setFont(new Font("고딕", Font.BOLD, 16));
                    topLabel.setForeground(Color.RED);
                }
                topLabel.setPreferredSize(new Dimension(60, 40)); // 일관된
크기로 유지한다
                topLabel.setMinimumSize(new Dimension(60, 40));
                topLabel.setMaximumSize(new Dimension(60, 40));
                topLabel.setHorizontalAlignment(SwingConstants.LEFT);
                itemPanel.add(topLabel);

                JLabel label = new JLabel(stack.get(i));
                label.setFont(new Font("고딕", Font.PLAIN, 16));
                Dimension labelSize = new Dimension(140, 40); // 라벨 크기 조정
                label.setPreferredSize(labelSize);
                label.setMinimumSize(labelSize);
                label.setMaximumSize(labelSize);
                label.setBorder(BorderFactory.createLineBorder(Color.BLACK));
                label.setHorizontalAlignment(SwingConstants.CENTER);
                label.setVerticalAlignment(SwingConstants.CENTER);
                label.setOpaque(true);

                if (i == top && blinkCounter == 1) {
                    label.setBackground(Color.YELLOW);
                } else {
                    label.setBackground(Color.WHITE);
                }

                itemPanel.add(label);
                stackPanel.add(itemPanel);
            }

            stackPanel.revalidate();
            stackPanel.repaint();
        }
```

파일 저장 및 불러오기:
현재 자료구조 상태를 텍스트 파일로 저장하고, 저장된 상태를 불러올 수 있는 기능을 제공합니다

<스택 상태를 파일에 저장하는 메서드>

```
private void saveStackToFile(String fileName) {
            try (BufferedWriter writer = new BufferedWriter(new
FileWriter(fileName))) {
                for (String value : stack) { // Stack의 모든 요소 저장한다
                    writer.write(value + "\n");
                }
                JOptionPane.showMessageDialog(this, "스택 상태가 저장되었습니다.",
"알림", JOptionPane.INFORMATION_MESSAGE);
            } catch (IOException e) {
                JOptionPane.showMessageDialog(this, "파일 저장 중 오류가
발생했습니다.", "오류", JOptionPane.ERROR_MESSAGE);
                e.printStackTrace();
            }
        }
```

<파일에서 스택 상태를 불러오는 메서드>

```
private void loadStackFromFile(String fileName) {
            try (BufferedReader reader = new BufferedReader(new
FileReader(fileName))) {
                stack.clear(); // 기존에 있던 스택을 초기화한다
                String line;
                while ((line = reader.readLine()) != null) {
                    stack.push(line); // 파일에서 읽은 데이터를 스택에 삽입한다
                }
                updateStackPanel(); // 스택을 시각화한다
                JOptionPane.showMessageDialog(this, "스택 상태가 불러와졌습니다.",
"알림", JOptionPane.INFORMATION_MESSAGE);
            } catch (IOException e) {
                JOptionPane.showMessageDialog(this, "파일 불러오기 중 오류가
발생했습니다.", "오류", JOptionPane.ERROR_MESSAGE);
                e.printStackTrace();
            }
        }
```

데이터 삽입시 반짝거리는 효과:
새로운 데이터가 추가되면 해당 데이터는 깜빡인다. 이는 스레드로 구현했다.
<추가된 데이터를 반짝이는 효과를 주는 메서드>

```
private void blinkStack() {
            // 이미 깜빡임 중이거나 깜빡임 대상 인덱스가 없으면 메서드를 종료한다
            if (isBlinking || top == -1) return;
            isBlinking = true; // 깜빡임 상태를 활성화시킨다

            // 별도의 스레드에서 깜빡임 동작 실행한다
            Thread blinkThread = new Thread(() -> {
                try {
                    // 총 6번 깜빡인다 (3번 점멸)
                    for (int i = 0; i < 6; i++) {
                        blinkCounter = i % 2; // 0과 1로 상태 번갈아 변경해
구현한다
                        updateStackPanel(); // 스택 패널 업데이트해 색을 반영한다.
                        Thread.sleep(300); // 300ms 간격으로 깜빡인다
                    }
                } catch (InterruptedException e) {
                    e.printStackTrace();
                } finally {
                    isBlinking = false; // 깜빡임 상태를 비활성화한다
                    blinkCounter = 0; // 깜빡임 상태를 초기화한다
                    top = -1; // 깜빡임 대상을 초기화한다
                    updateStackPanel(); // 스택 패널을 최종 업데이트한다
                }
            });
            blinkThread.start(); // 스레드를 시작한다
        }
```

큐 시뮬레이션:
JPanel클래스를 상속받아 그래픽으로 데이터가 들어갈 공간을 구현했다.
사용자 입력으로 데이터를 받아서 배열에 저장한다. 그 후 배열을 읽어 표시한다.
<큐를 시각적으로 표현하는 메서드>

```
class CircleCanvas extends JPanel {

            public CircleCanvas() {
                setBackground(Color.WHITE); // 캔버스 배경을 흰색으로 설정한다
            }

            @Override
```

```
            protected void paintComponent(Graphics g) {
                super.paintComponent(g); // 부모 클래스의 paint 메서드 호출한다
                Graphics2D g2 = (Graphics2D) g; // 업캐스팅을 해 안티앨리어싱을
한다
                // 그래픽 품질 향상을 위한 안티앨리어싱을 설정한다
                g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

                // 원형 큐를 그리기 위한 중심 좌표와 반지름을 계산한다
                int width = getWidth(); // 패널의 가로
                int height = getHeight();

                int centerX = width / 2; // 패널의 중심 좌표
                int centerY = height / 2;
                int radius = Math.min(width, height) / 3; // 패널 크기에 따라
반지름이 동적으로 조정한다

                Font font = new Font("SansSerif", Font.PLAIN, 14); // 기본 폰트를
설정한다
                g2.setFont(font);
                FontMetrics fm = g2.getFontMetrics(); // 텍스트의 너비와 높이를
계산한다 (텍스트 가운데 정렬하기위해)

                // 큐의 각 요소를 원형 배치로 그린다
                for (int i = 0; i < MAX_SIZE; i++) {
                    // 요소의 인덱스에 따라 위치를 계산한다
                    double angle = Math.toRadians((360.0 / MAX_SIZE) * i - 90);
// 12시 방향을 시작점으로 지정한다
                    int x = (int) (centerX + radius * Math.cos(angle));
                    int y = (int) (centerY + radius * Math.sin(angle));

                    // 새로 삽입된 데이터(rear)만 깜빡임 처리와 색을 지정한다
                    if (i == rear && front != rear) {
                        if (blinkCounter == 1) {
                            g2.setColor(Color.YELLOW); // 깜빡이는 색상
                        } else {
                            g2.setColor(Color.WHITE); // 통일된 기본 색상
                        }
                    } else if (queue[i] != null) {
                        g2.setColor(Color.WHITE); // 통일된 기본 색상
                    } else {
```

```
                    g2.setColor(Color.GRAY); // 비어 있는 경우
                }

                // 원형 슬롯 그리기
                g2.fillOval(x - 35, y - 30, 60, 60); // 슬롯 내부 색상 칠하기
                g2.setColor(Color.BLACK);
                g2.drawOval(x - 35, y - 30, 60, 60); // 슬롯 외곽선 그리기

                // 텍스트 출력 (가운데 정렬 및 길이 조정)
                if (queue[i] != null) {
                    String text = queue[i];
                    if (fm.stringWidth(text) > 50) { // 텍스트 길이가 원을
넘으면 자르기
                        text = text.substring(0, Math.min(text.length(), 5)) +
"...";
                    }
                    int textWidth = fm.stringWidth(text);
                    int textHeight = fm.getAscent();
                    g2.drawString(text, x - textWidth / 2 - 4, y + textHeight
/ 4 + 2);
                }

                // 인덱스 위치에 "FRONT" 또는 "REAR" 라벨을 추가한다
                if (i == front) {
                    g2.setColor(Color.BLUE);
                    g2.drawString("FRONT: " + i, x - 30, y - 40);
                }
                if (i == rear && queue[i] != null) {
                    g2.setColor(Color.RED);
                    g2.drawString("REAR: " + i, x - 30, y + 55);
                }
            }
        }
    }
```

파일 저장 및 불러오기:
현재 자료구조 상태를 텍스트 파일로 저장하고, 저장된 상태를 불러올 수 있는 기능을 제공한다. 비어있는 부분은 'null'로 저장하고 rear와 front 값도 같이 저장한다.
<큐 상태를 파일에 저장하는 메서드>

```
private void saveQueueToFile(String fileName) {
        try (BufferedWriter writer = new BufferedWriter(new
```

```
FileWriter(fileName))) {
                for (int i = 0; i < MAX_SIZE; i++) {
                    if (queue[i] != null) {
                        writer.write(queue[i] + "\n"); // 각 데이터를 파일에
저장한다
                    } else {
                        writer.write("null\n"); // 비어있는 부분을 null로 저장한다
                    }
                }
                writer.write(front + "\n"); // front 값을 저장한다
                writer.write(rear + "\n");  // rear 값을 저장한다
                JOptionPane.showMessageDialog(this, "큐 상태가 저장되었습니다.",
"알림", JOptionPane.INFORMATION_MESSAGE);
            } catch (IOException e) {
                JOptionPane.showMessageDialog(this, "파일 저장 중 오류가
발생했습니다.", "오류", JOptionPane.ERROR_MESSAGE);
                e.printStackTrace();
            }
        }
```

<파일에서 큐 상태를 불러오는 메서드>

```
private void loadQueueFromFile(String fileName) {
            try (BufferedReader reader = new BufferedReader(new
FileReader(fileName))) {
                for (int i = 0; i < MAX_SIZE; i++) {
                    String line = reader.readLine();
                    if ("null".equals(line)) {
                        queue[i] = null; // 비어있는 부분을 복원한다
                    } else {
                        queue[i] = line; // 큐 데이터를 복원한다
                    }
                }
                front = Integer.parseInt(reader.readLine()); // front 값 불러오기
                rear = Integer.parseInt(reader.readLine());  // rear 값 불러오기

                canvas.repaint(); // 큐 시각화 갱신
                JOptionPane.showMessageDialog(this, "큐 상태가 불러와졌습니다.",
"알림", JOptionPane.INFORMATION_MESSAGE);
            } catch (IOException | NumberFormatException e) {
                JOptionPane.showMessageDialog(this, "파일 불러오기 중 오류가
발생했습니다.", "오류", JOptionPane.ERROR_MESSAGE);
```

```
                e.printStackTrace();
            }
        }
```

데이터 삽입시 반짝거리는 효과:
새로운 데이터가 추가되면 해당 데이터는 깜빡인다. 이는 스레드로 구현했다.
<추가된 데이터를 반짝이는 효과를 주는 메서드>

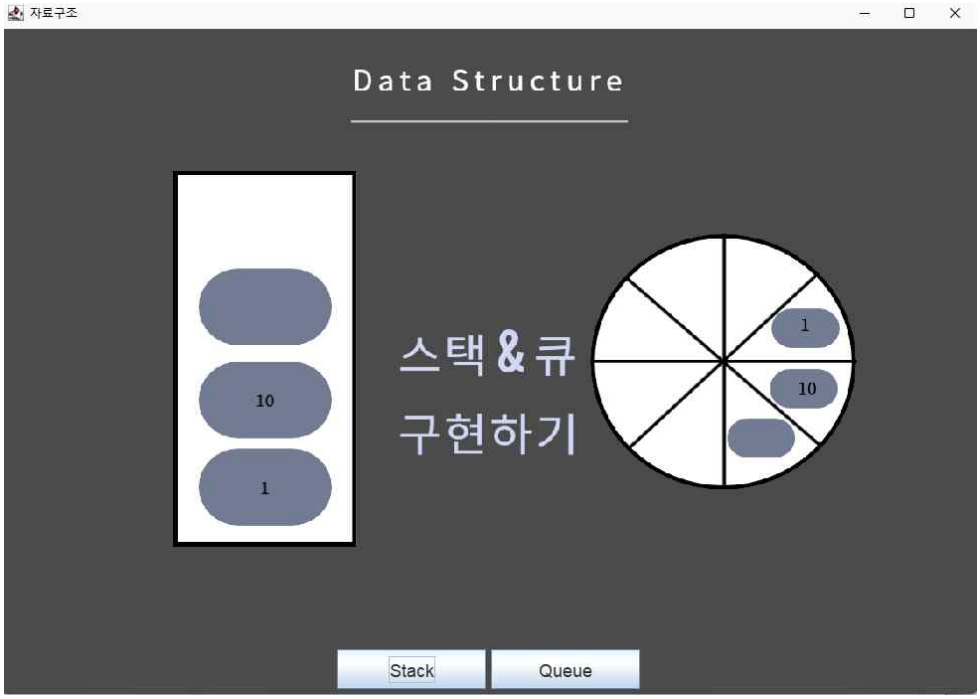
```
private void blinkQueue() {
            // 이미 깜빡임이 진행 중이거나 깜빡일 대상이 없으면 메서드를 종료한다
            if (isBlinking || rear == -1) return;
            isBlinking = true;

            // 깜빡임 애니메이션 처리를 위한 스레드를 시작한다
            Thread blinkThread = new Thread(() -> {
                try {
                    for (int i = 0; i < 6; i++) { // 6번 깜빡임 (3번 반복)
                        blinkCounter = i % 2; // 0(꺼짐)과 1(켜짐) 상태를 번갈아
변경한다
                        canvas.repaint(); // 캔버스를 다시 그려 상태를 반영한다
                        Thread.sleep(300); // 깜빡임 간격 300ms로 설정한다
                    }
                } catch (InterruptedException e) {
                    e.printStackTrace();
                } finally {
                    // 깜빡임 상태 초기화 및 캔버스를 다시 그려 원래 상태로
복원한다
                    isBlinking = false;
                    blinkCounter = 0; // 깜빡임 카운터을 초기화한다
                    canvas.repaint(); // 최종 상태를 화면에 반영한다
                }
            });
            blinkThread.start(); // 스레드를 실행한다
        }
```

6. 앱 메뉴얼

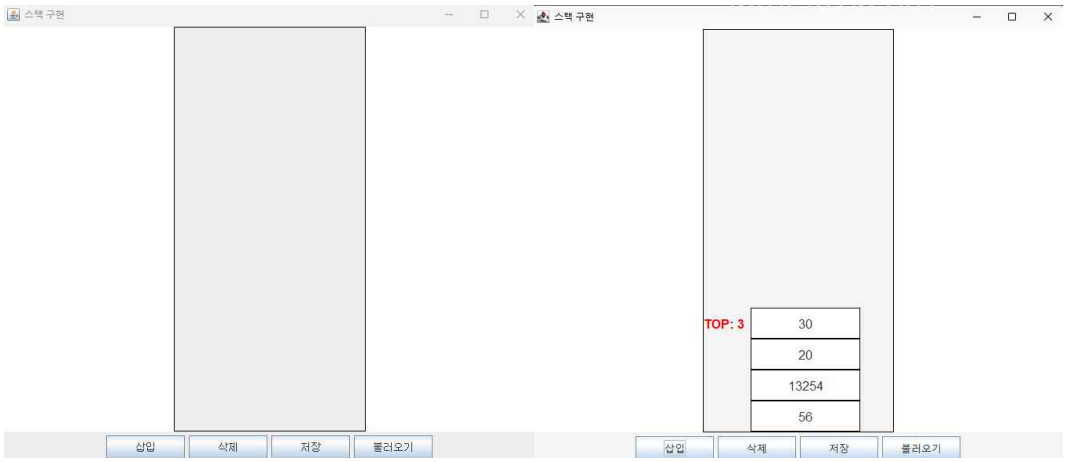
6.1 프로그램 사용법

<메인 화면>



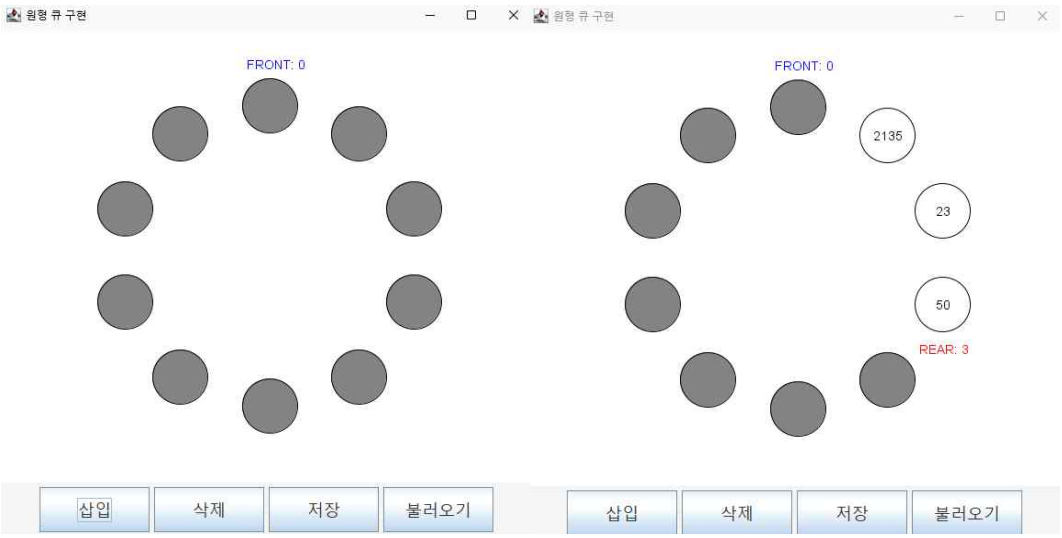
사용자는 "Stack" 또는 "Queue" 버튼을 클릭하여 시뮬레이션 화면으로 이동한다.

<Stack 화면>



삽입 버튼: 데이터를 입력하고 삽입 버튼을 눌러 데이터를 추가한다.
삭제 버튼: 삭제 버튼을 눌러 데이터를 제거한다.
저장 버튼: 현재 스택의 상태를 파일로 저장한다.
불러오기 버튼: 저장된 상태를 불러온다.

<Queue 화면>



삽입 버튼: 데이터를 입력하고 삽입 버튼을 눌러 데이터를 추가한다.
삭제 버튼: 삭제 버튼을 눌러 데이터를 제거한다.
저장 버튼: 현재 큐의 상태를 파일로 저장한다.
불러오기 버튼: 저장된 상태를 불러온다.

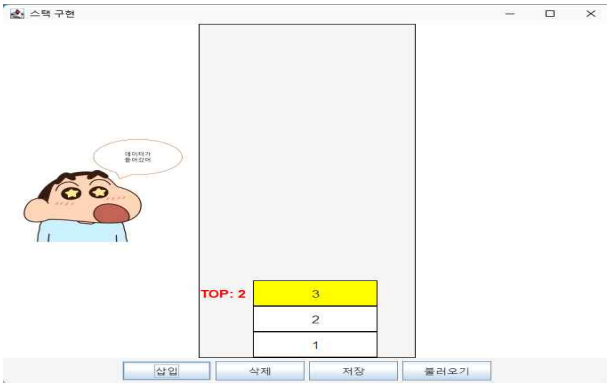
6.2 진행 과정

6.2.1 스택

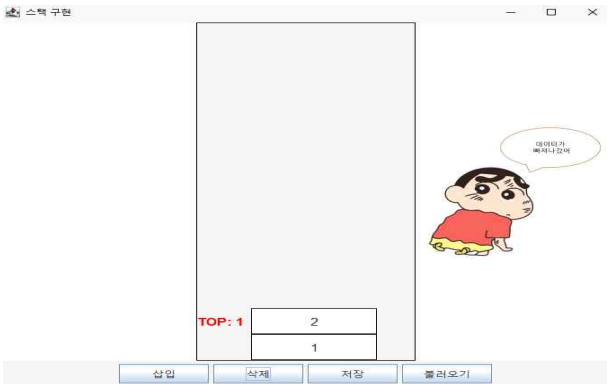
PUSH:

하단에 삽입 버튼을 누르면 데이터를 입력할 수 있게 팝업 창이 뜬다. 이 곳에 사용자가 원하는 데이터 값을 저장한다. 실시간으로 추가되는 모습을 볼 수 있다. 데이터가 길어지면 '...'으로 생략된다.

이미지 및 반짝임:
삽입된 데이터는 3번 깜빡임으로 강조가 되고 화면 왼쪽에 이미지가 2초 동안 표시된다.

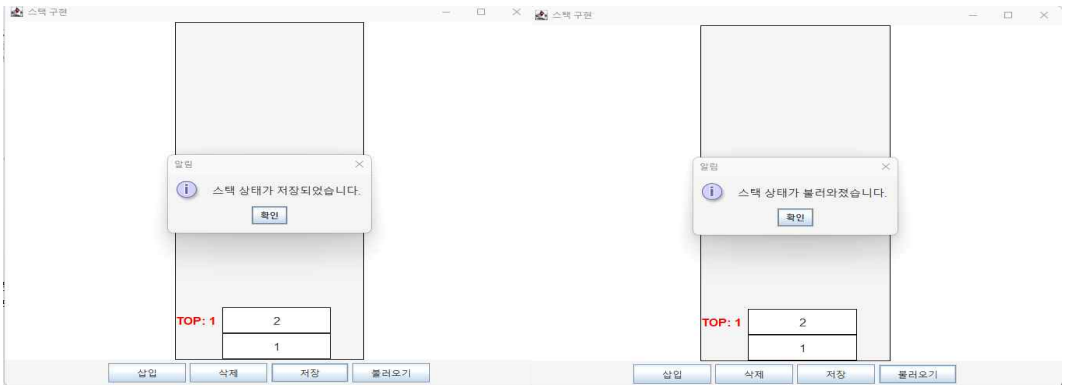


POP:
하단에 삭제 버튼을 누르면 스택 맨 위의 데이터가 삭제된다. 또한 화면 오른쪽 이미지가 2초 동안 표시된다.



경고 메시지:
스택이 비었을 경우 삭제 버튼을 누를 경우와 스택이 가득 찼을 때 삽입 버튼을 누른 경우 경고음을 통해 위와 같이 알림이 표시된다.

저장 및 불러오기:
하단에 저장 버튼을 누르면 현재 스택의 상태를 저장하고 위와 같이 알림 메시지가 뜬다.
마찬가지로 하단에 불러오기 버튼을 누르면 저장된 스택을 불러온 후 알림 메시지가 뜬다.
이를 통해 진행했던 작업을 이어서 할 수 있다.



<stack_state.txt> 저장 상태

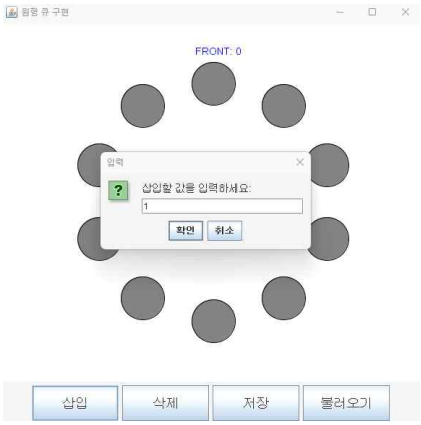
6.2.2 큐

Enqueue:

하단에 삽입 버튼을 누르면 데이터를 입력할 수 있게 팝업 창이 뜬다.

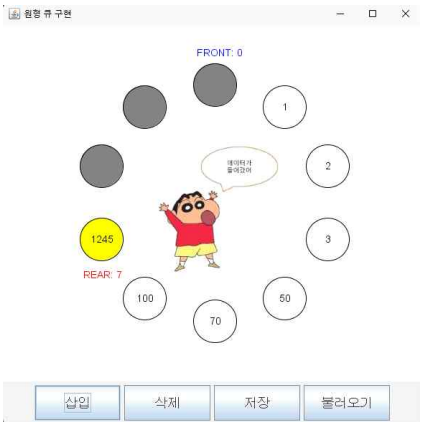
이 곳에 사용자가 원하는 데이터 값을 저장한다. 실시간으로 추가되는 모습을 볼 수 있다.

데이터가 길어지면 '...'으로 생략된다.



이미지 및 반짝임:

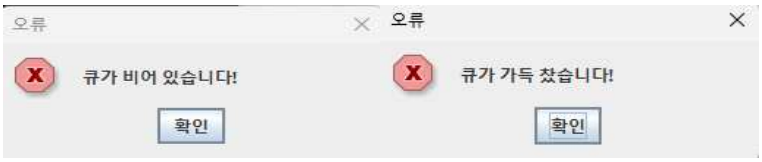
삽입된 데이터는 3번 깜빡임으로 강조되고 화면 가운데에 이미지가 1초 동안 표시된다.



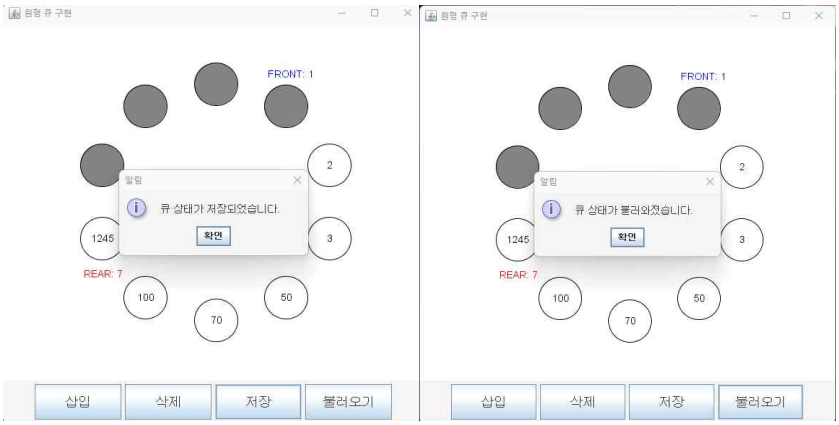
Dequeue:

하단에 삭제 버튼을 누르면 front의 값이 삭제된다. 또한 화면 가운데에 이미지가 1초 동안 표시된다.

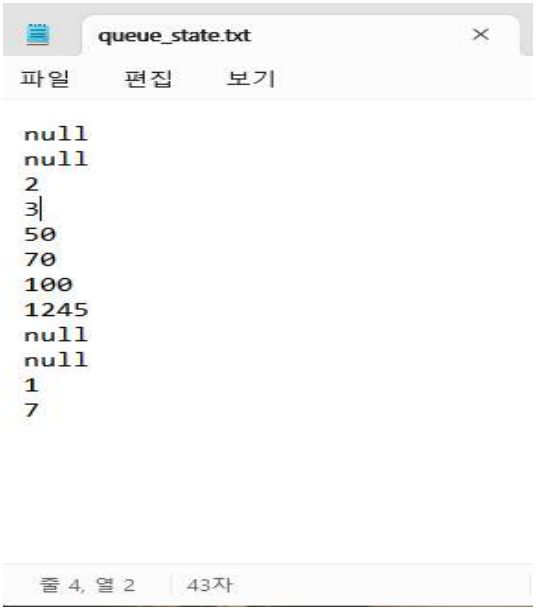
경고메시지:
큐가 비었을 경우 삭제 버튼을 누를 경우와 큐가 가득 찼을 때 삽입 버튼을 누른 경우 경고음을 통해 위와 같이 알림이 표시된다.



저장 및 불러오기:
하단에 저장 버튼을 누르면 현재 큐의 상태를 저장하고 위와 같이 알림 메시지가 뜬다.
마찬가지로 하단에 불러오기 버튼을 누르면 저장된 큐를 불러온 후 알림 메시지가 뜬다.
이를 통해 진행했던 작업을 이어서 할 수 있다.



<queue_state.txt> 저장 상태



7. Good / Bad 분석

Good

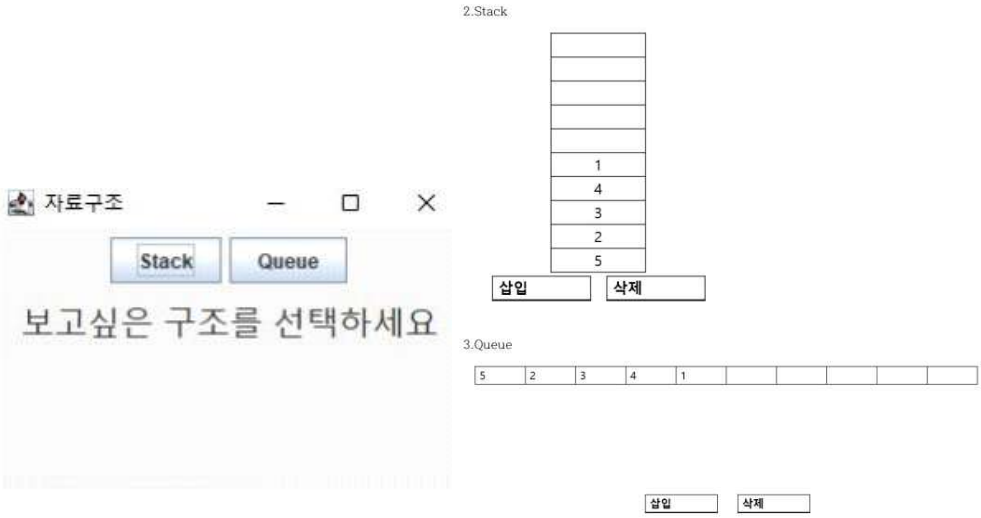
- 직관적인 인터페이스와 시각적 효과로 누구나 쉽게 사용할 수 있다.
- 자료구조의 동작 과정을 실시간으로 확인할 수 있어 학습에 유용하다.
- 설계가 모듈화되어 다른 자료구조를 추가하기 쉽다.
- 데이터를 저장하고 다시 불러올 수 있는 기능으로 활용도를 높였다.

Bad

- 현재는 스택과 큐만 지원하며, 다른 자료구조의 추가가 필요하다.
- 애니메이션이 간단하여 복잡한 자료구조에는 적합하지 않을 수 있다.
- 디자인이 단순하여 사용자 경험을 더 향상시킬 여지가 있다.
- 사이즈가 제한적이다.

8. 프로젝트 진행 과정

먼저 팀원과 주제 관련 토의를 진행했습니다. 자료구조 시간에 배운 스택과 큐를 이해하기 쉽게 만드는 프로그램을 만들어보자고 결정했습니다. 아래는 초기의 메인 화면과 스택과 큐 화면입니다.



스택과 큐 모두 최대 사이즈는 10으로 지정을 했고 경고음성을 사이즈보다 하나 적을 때 출력하기로 했지만, 더 이상 추가할 수 없거나 삭제할 수 없을 경우에 경고음성을 출력하기로 고쳤습니다. 이렇게 정해두고 팀원은 스택쪽을 구현하기로 하였고 나는 큐 쪽을 담당하기로 하였습니다. 위에 사진과 같이 저렇게 큐를 표현하니 뭔가 부족하다고 느껴 그래픽을 이용해서 구현해보기로 하였습니다. 그랬더니 문제점이 데이터를 저장하는 슬롯들을 배치하는 것이 힘들어 chatGPT를 이용해 구현을 부탁했습니다.

그리고 부가적으로 깜빡임, 이미지 타이머, 파일 입출력 처리 후 팀원과 코드를 합쳐 코드를 완성시켰습니다. 그 후 음성 출력 클래스를 만들며 완성시켰습니다.

9. 결론

미니프로젝트를 통해 실제 구현, 테스트를 거치면서 디버깅 과정이 얼마나 중요한지 느꼈습니다. 오류가 발생했을 때, 문제의 원인을 정확히 파악하는 것이 얼마나 중요한지 실감하게 되었습니다. 또한 chatGPT는 도구로써 사용하는게 맞다고 느꼈습니다. 내가 원하는 코드를 짜주긴하지만 완벽하지않고 바라던 코드와는 조금은 틀리기 때문에 그렇게 느꼈습니다.
그리고 대학교 처음으로 팀프로젝트를 진행봤는데 git의 중요함도 느꼈습니다.

프로젝트를 하면서 코드를 깔끔하고 효율적이게 작성하는것의 중요성을 느꼈고, 아직 많이 부족하다고 깨달았습니다. 훌륭한 컴퓨터 전문가로서 성장하기 위해 꾸준히 최신 기술을 학습하고 다양한 문제 해결 경험을 쌓아야 함을 알게 되었습니다.