Homework 4 보고서 **18013171 김리아**

<구현 순서>

1) Frame 생성: JavaHW4 클래스
2) Panel 생성: Hw4Panel 클래스
3) 버튼이 들어갈 Panel 생성: Hw4ButtonPanel 클래스
4) Panel 비율 설정: Hw4Panel 클래스
5) 버튼 꾸미기: Hw4Button 클래스
6) 라벨이 들어갈 Panel 생성 및 꾸미기: Hw4LabelPanel 클래스
7) 라벨 꾸미기: Hw4Label 클래스
8) 계산기 기능 구현: Hw4Panel 클래스

1. Frame 생성: JavaHW4 클래스

1) public 클래스에서 JFrame 클래스를 상속받고, main 함수에서 JavaHW4 클래스를 객체화한다.

```
public class JavaHW4 extends JFrame {
   public static void main(String[] args){ new JavaHW4(); }
}
```

2) 생성자 안에서 Frame을 생성하고, Frame의 배경색을 검은색으로 지정한다.

```
JavaHW4() { getContentPane().setBackground(Color.balck); }
```

*** 시행착오**

- Frame에 검은색 배경을 설정하기 위해 setBackground 메소드를 사용했더니 적용되지 않았다. getContentPane().setBackground()로 코드를 변경했다.

*** 배운 점**

- getContentPane 메소드로 컨테이너를 불러온 후에 setBackground 메소드로 배경색을 설정해야 Frame의 배경색을 변경할 수 있다.

2. Panel 생성: Hw4Panel 클래스

1) Hw4Panel 클래스를 생성하고, JPanel 클래스를 상속받는다.
2) 라벨과 버튼이 들어갈 Panel의 인스턴스 변수를 각각 Hw4LabelPanel, Hw4ButtonPanel 타입으로 선언한다.
3) 라벨과 16개의 버튼이 저장될 배열을 인스턴스 변수로 선언한다.

4) 생성자 안에서 인스턴스 변수로 선언한 Panel과 Component를 생성한다.

```
Hw4Panel() {
   lp = new Hw4LabelPanel();
   bp = new Hw4ButtonPanel();
   label = new JLabel("0");
   btns = new JButton[16];
}
```

5) 생성한 label을 Hw4LabelPanel에 추가한다.

```
p.add(label);
```

6) 버튼의 각 이름을 String 배열로 생성하고, 각 버튼을 생성해 Hw4ButtonPanel에 추가한다.

```
String btnTexts[] = { "7", "8", "9", "C", "4", "5", "6", "+", "1", "2", "3", "-", "0", " ", " ", "=" };
for(int i=0; i<16; i++) {
   btns[i] = new JButton(btnTexts[i]);
   bp.add(btns[i]);
}
```

7) 두 개의 Panel을 Hw4Panel에 추가하고, Hw4Panel을 JavaHW4 클래스에서 Frame에 추가한다.

*** 시행착오**

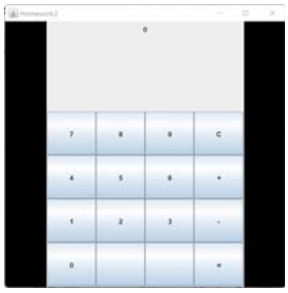
- 라벨과 버튼을 각각 Hw4LabelPanel과 Hw4ButtonPanel 클래스의 생성자 안에서 생성하고 추가했었다. 하지만 버튼을 클릭한 결과를 라벨에 반영하기에 어려움이 있었다. Hw4Panel에서 컴포넌트를 생성해 각 Panel에 추가하고, 인스턴스 변수를 이용해 Panel과 컴포넌트에 접근하는 방식으로 변경했다.

3. 버튼이 들어갈 Panel 생성: Hw4ButtonPanel 클래스

1) Hw4ButtonPanel 클래스를 생성하고, JPanel 클래스를 상속받는다.
2) 생성자 안에서 레이아웃 매니저를 4*4의 GridLayout으로 변경한다.

```
setLayout(new GridLayout(4, 4));
```

4. Panel 비율 설정: Hw4Panel 클래스



1) Hw4Panel 클래스에서 ComponentListener 인터페이스를 상속받고, 해당 인터페이스의 모든 추상 메소드를 오버라이드한다.
2) 화면의 크기가 변경될 때 라벨과 버튼 Panel의 비율을 조절하기 위해, ComponentListenr의 추상 메소드 중 componentResized 메소드를 구현한다.
3) Hw4Panel의 가로세로 비율과, 라벨 Panel과 버튼 Panel의 세로 길이 비율을 변수로 선언한다.

```
double fRatio = 1.35; // 가로 370*세로 500
double pRatio = 0.34; // 1:2 (라벨:버튼 Panel 세로 길이)
```

4) Frame의 크기를 구하고, 프레임의 가로세로 비율에 맞게 Panel의 크기를 계산한다. 계산한 세로 길이가 Frame의 세로 길이보다 길다면, 값을 조정한다.

```
Dimension d = getParent().getSize();
int w = d.width;
int h = (int)(w*fRatio);
if(h > d.height) {
   h = d.height;
   w = (int)(h/fRatio);
}
setBounds(d.width/2-w/2, d.height/2-h/2, w, h);
```

5) 라벨과 버튼 Panel의 크기를 새로 계산한 값에 맞춰 조정한다.

```
lp.setPreferredSize(new Dimension(w, (int)(h*pRatio)));
lp.setBounds(0, 0, w, (int)(h*pRatio));
bp.setPreferredSize(new Dimension(w, (int)(h*(1-pRatio))));
bp.setBounds(0, (int)(h*pRatio), w, (int)(h*(1-pRatio)));
```

6) 라벨의 영역의 크기를 라벨 Panel의 크기와 똑같이 설정한다.

```
label.setPreferredSize(lp.getPreferredSize());
label.setBounds(lp.getBounds());
```

7) Hw4Panel 생성자에서 ComponentListener를 등록한다.

```
addComponentListener(this);
```

*** 시행착오**

- HW4Panel의 레이아웃 매니저를 BorderLayout으로 설정하고, 라벨과 버튼 Panel을 각각 NORTH, CENTER에 배치해 내부 패널의 크기가 제대로 변경되지 않았다. 레이아웃 매니저를 설정하는 코드를 지우고, Panel의 기본 레이아웃 매니저인 FlowLayout으로 구현했다.

- getSize 메소드로 Panel의 크기를 구하고, 비율을 변경하려고 했었다. 하지만 프레임의 크기에 따라 비율이 변경되어야 하므로 getParent().getSize()로 코드를 변경했다.

*** 배운 점**

- ComponentListener를 상속받고, componentResized 메소드를 통해 화면 크기가 바뀔 때마다 비율에 맞게 Panel의 크기를 조절할 수 있다.
- Panel 안에서 getParent().getSize()를 통해 프레임의 크기를 구할 수 있다.
- Panel의 크기를 변경할 땐 setSize 메소드가 아닌, setPreferredSize 메소드를 사용해야 한다.
- setBounds 메소드를 통해 컴포넌트가 보여지는 위치와 크기를 지정할 수 있다.

5. 버튼 꾸미기: Hw4Button 클래스

1) Hw4Button 클래스를 생성하고, JButton 클래스를 상속받는다.
2) 버튼의 문자열 색상을 인스턴스 변수로 선언한다. 버튼의 문자열과 문자열의 색상 값을 설정하기 위한 생성자를 작성한다.

```
private String color;
Hw4Button(String s, Color c) { super(s); color=c; }
```

3) 버튼을 꾸미기 위해 paintComponent 메소드를 오버라이드한다.
4) 버튼 영역의 크기를 구하고, 둥근 사각형으로 버튼의 배경을 그린다.

```
Dimension d = getSize();
int p = d.width/12;
g2.setColor(new Color(230, 220, 200));
g2.fillRoundRect(p, p, d.width-(p*2), d.height-(p*2), p, p);
```

5) 버튼에 그릴 text의 폰트를 설정한다.

```
g2.setFont(new Font("Arial", Font.BOLD, (int)(d.height/1.6)));
```

6) 버튼 text의 그림자를 흰색으로 먼저 그린 후, 버튼 text를 설정된 색상으로 그린다. 그림자는 text 보다 살짝 왼쪽 아래에 그려지게 한다.

```
g2.setColor(Color.white);
g2.drawString(getText(), (int)(d.width/2-(p*1.7)-1), (int)(d.height/2+(p*2.3)+2));
g2.setColor(color);
g2.drawString(getText(), (int)(d.width/2-(p*1.7)), (int)(d.height/2+(p*2.3)));
```

7) Hw4Panel 클래스에서 버튼을 생성하는 코드를 수정한다. 버튼의 이름이 존재하지 않는 경우 기본 디자인으로 표시될 수 있게 JButton 타입으로, 이름이 존재하는 경우 Hw4Button 타입으로 생성한다. 리셋 버튼은 붉은색으로, 숫자와 연산자 버튼은 갈색으로 설정한다.

```
for(int i=0; i<16; i++) {
   if(btnTexts[i].equals(" ")) btns[i] = new JButton();
   else if (btnTexts[i].equals("C")) btns[i] = new Hw4Button(btnTexts[i], new Color(200, 120, 100));
   else btns[i] = new Hw4Button(btnTexts[i], new Color(130, 120, 100));
   bp.add(btns[i]);
}
```

*** 시행착오**

- 버튼 배열과 모든 버튼의 타입을 Hw4Button으로 변경해 모든 버튼에 둥근 사각형의 배경이 추가되었다. 버튼 배열을 JButton 타입으로 선언하고, 버튼의 이름이 없는 경우에는 JButton 타입으로 생성해 기본 디자인 그대로 추가될 수 있도록 변경했다.
- Hw4Panel 클래스에서 버튼을 추가할 때 setColor로 버튼 문자열의 색상을 설정했었다. 생성자를 통해 각 버튼의 이름과 색상을 설정하는 방식으로 변경했다.
- 버튼의 배경과 폰트의 크기 및 위치는 프레임의 크기에 따라 변경될 수 있도록 직접 실행해보면서 계산값을 조절했다.

6. 라벨이 들어갈 Panel 생성 및 꾸미기: Hw4LabelPanel 클래스

1) Hw4LabelPanel 클래스를 생성하고, JPanel 클래스를 상속받는다.
2) Panel의 배경을 꾸미기 위해 paintComponent 메소드를 오버라이드한다.
3) Hw4LabelPanel 영역의 크기를 getSize 메소드를 이용해 구한다.
4) 흰색~연한 회색의 첫 번째 세로 그라데이션을 Hw4LabelPanel 영역에 꽉 채워서 그린다.

```
GradientPaint gp = new GradientPaint(0, 0, Color.white, 0, d.height, Color.lightGray);
g2.setPaint(gp);
g2.fillRect(0, 0, d.width, d.height);
```

5) 연한 회색~진한 회색의 두 번째 세로 그라데이션을 상하좌우 3씩 띄워서 사각형으로 중앙에 그린다.

```
Color c1 = new Color(150, 150, 150);
Color c2 = new Color(100, 100, 100);
gp = new GradientPaint(0, 0, c1, 0, d.height, c2);
g2.setPaint(gp);
g2.fillRect(3, 3, d.width-6, d.height-6);
```

6) 진한 초록색~연한 초록색의 세 번째 세로 그라데이션을 사각형으로 화면 크기에 맞춰 중앙에 그린다.

```
c1 = new Color(100, 110, 100);
c2 = new Color(150, 160, 150);
gp = new GradientPaint(0, 0, c1, 0, d.height, c2);
g2.setPaint(gp);
g2.fillRect(d.width/100+p, d.width/100+p, d.width-(d.width/50)-(p*2), d.height-(d.width/50)-(p*2));
```

*** 시행착오**

- 그라데이션 사각형 그리는 것을 Hw4Label 클래스에서 구현했었지만, Panel의 배경색을 꾸미는 것이므로 Hw4LabelPanel로 코드를 옮겼다.

7. 라벨 꾸미기: Hw4Label 클래스

1) Hw4Label 클래스를 생성하고, JLabel 클래스를 상속받는다.
2) 인스턴스 변수로 라벨의 문자열이 저장될 변수를 선언하고, 생성자와 setLabel 메소드를 통해 변수의 값을 설정할 수 있게 한다.

```
String label;
Hw4Label(String s) { label = s; }
void setLabel(String s) { label = s; }
```

3) 라벨을 꾸미기 위해 paintComponent 메소드를 오버라이드한다.
4) 라벨 영역의 크기를 getSize 메소드를 이용해 구한다.
5) 폰트를 설정하고, 해당 폰트로 라벨의 문자열을 그렸을 때 가로, 세로 길이를 구한다.

```
Font font = new Font("Arial", Font.PLAIN, d.width/5);
g2.setFont(font);
int lw = (int) font.getStringBounds(label, g2.getFontRenderContext()).getWidth();
int lh = (int) font.getStringBounds(label, g2.getFontRenderContext()).getHeight();
```

6) 라벨의 그림자를 흰색으로 먼저 그린 후, 문자열을 검은색으로 오른쪽 정렬해 그린다. 라벨은 초록색 배경 사각형보다 안쪽에 그리고, 그림자는 문자열보다 살짝 아래에 그린다.

```
int p = d.width/20 + 15;
g2.setColor(Color.white);
g2.drawString(label, d.width-lw-p, d.height/2+lh/3+1);
g2.setColor(Color.white);
g2.drawString(label, d.width-lw-p, d.height/2+lh/3+1);
```

7) Hw4Panel 클래스에서 JLabel 타입으로 선언된 인스턴스 변수를 Hw4Label 타입으로 변경한다.

*** 시행착오**

- Label로 그려야 할 문자열을 setText로 설정해주었더니, 기본 문자열과 drawString 메소드 둘 다 그려지는 문제가 있었다. 인스턴스 변수에 문자열을 저장해두고, drawString으로만 그려질 수 있도록 변경했다.
- 라벨을 오른쪽으로 정렬하기 위해, 출력 위치를 문자열의 길이에 임의의 너비를 곱해 지정해주었다. 하지만 숫자마다 글자의 너비가 다르고, 화면의 크기가 바뀌면 제대로 정렬되지 않았다. drawString 메소드로 그리는 문자열의 너비를 구하는 함수를 찾아서 코드를 변경해주었다.
- 라벨을 Panel의 수직 중앙에 그리기 위해 직접 실행해보면서 계산값을 조절했다.

*** 배운 점**

- getStringBounds 메소드를 이용해 drawString 메소드로 그리는 문자열의 크기를 알 수 있다.

8. 계산 기능 구현: Hw4Panel 클래스

50-60+100=90을 순서대로 실행한 결과

1) 버튼 이벤트 처리를 위해 Hw4Panel 클래스에서 ActionListener 인터페이스를 상속받고, 해당 인터페이스의 추상 메소드 actionPerformed를 오버라이드한다.
2) 계산기 버튼이 눌린 순서대로 문자열을 저장할 수 있는 LinkedList 타입의 변수와 숫자 문자열을 저장할 변수, 계산 결과를 저장할 변수를 인스턴스 변수로 선언한다.

```
LinkedList<String> cal = new LinkedList<>();
String num = "";
int result = 0;
```

3) actionPerformed 메소드 안에서 클릭된 버튼의 문자를 알아낸다.

```
String in = ((JButton) e.getSource()).getText();
```

4) 클릭한 버튼이 리셋(C) 버튼이라면, 리스트와 숫자 문자열, 계산 결과를 모두 초기화하고, 라벨의 값을 0으로 설정한다.

```
cal.clear();
num = "";
result = 0;
label.setLabel(result+"");
```

5) 클릭한 버튼이 숫자라면, 문자열 변수 num에 이어 붙인다.

```
num += in;
label.setLabel(result+"");
```

6) 클릭한 버튼이 연산(+, -, =) 버튼이라면, num 변수에 저장된 문자열을 리스트에 추가하고, 리스트에 저장된 문자를 하나씩 꺼내 계산한다. 꺼낸 문자가 +, -의 연산자인 경우, op 변수에 저장하고 숫자인 경우, op에 저장된 변수에 따라 계산을 수행해 result 변수에 저장한다.

```
if(num.equals("") == false) {
   cal.add(num);
   num = "";
}
if(cal.size() == 0) result = 0;
else result = Integer.parseInt(cal.get(0));
for(int i=1; i<cal.size(); i++) {
   if(cal.get(i).equals("+") || cal.get(i).equals("-"))
      op = cal.get(i);
   else {
      if(op.equals("+")) result += Integer.parseInt(cal.get(i));
      else result -= Integer.parseInt(cal.get(i));
   }
}
```

7) 계산이 완료되면 리스트를 초기화한다. 마지막 연산 결과만 리스트에 추가하고, 라벨에 표시한다.

```
cal.clear();
cal.add(result+"");
label.setLabel(result+"");
```

8) 마지막에 클릭한 버튼이 +, - 버튼이라면, 계속 연산을 수행하기 위해 해당 연산자 문자도 리스트에 추가한다.

```
if(in.equals("+") || in.equals("-")) cal.add(in);
```

9) repaint() 함수를 호출해 계산 결과를 라벨 영역에 그린다.

10) Hw4Panel에서 버튼을 생성할 때, 이름이 있는 버튼에만 ActionListener를 등록한다.

```
if(btnTexts[i].equals(" "))
   btns[i] = new JButton();
else if (btnTexts[i].equals("C")) {
   btns[i] = new Hw4Button(btnTexts[i], new Color(200, 120, 100));
   btns[i].addActionListener(this);
} else {
   btns[i] = new Hw4Button(btnTexts[i], new Color(130, 120, 100));
   btns[i].addActionListener(this);
}
```

*** 시행착오**

- num1, num2, op 세 가지 변수를 선언하고 버튼을 클릭할 때마다 숫자 버튼인 경우, 문자를 숫자로 변환해 계산하는 방식으로 구현했다. 하지만 연속으로 계산을 수행하는 과정에서 코드가 복잡해져서 LinkedList에 문자로 추가해두고, +, -, = 연산자가 입력되었을 때만 계산하는 방식으로 변경했다.
- "=" 버튼이 눌릴 때만 라벨에 계산 결과를 표시하도록 코드를 작성해, 계산을 연속으로 수행하는 경우에도 마지막 결과만 라벨에 보여졌다. "+"와 "-" 버튼이 눌리면 중간 계산 결과를 라벨에 보여줄 수 있도록 코드를 수정했다.