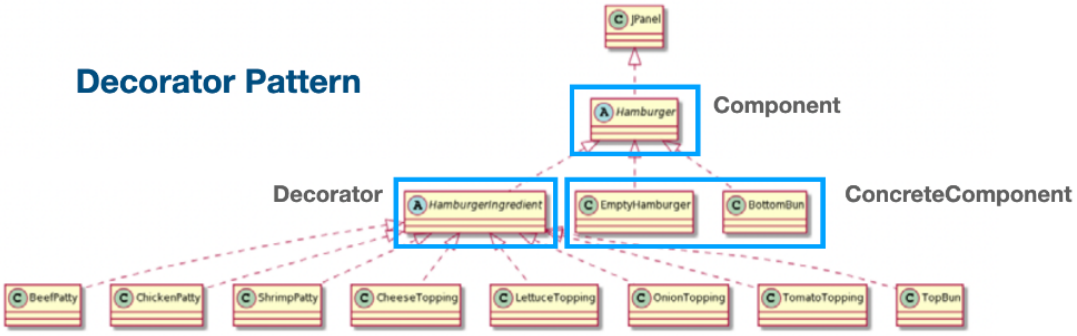
Java Programming II Lab 4

교 과 명 : 자바프로그래밍 2
담당교수명 : 박경신 교수님
학 과 : 컴퓨터공학과
학 번 : 32185010
성 명 : 홍찬희
제 출 일 : 2021. 10. 07

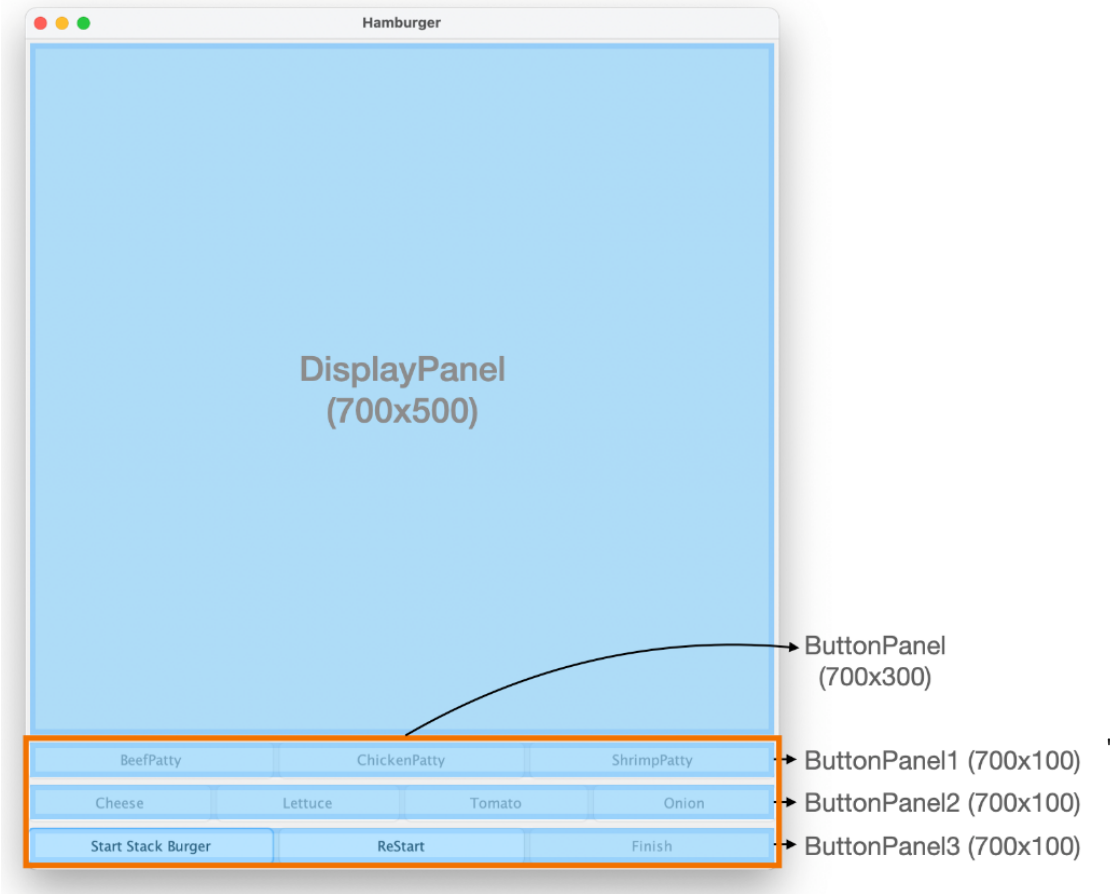
DKU
단국대학교
DANKOOK UNIVERSITY

Lab 4

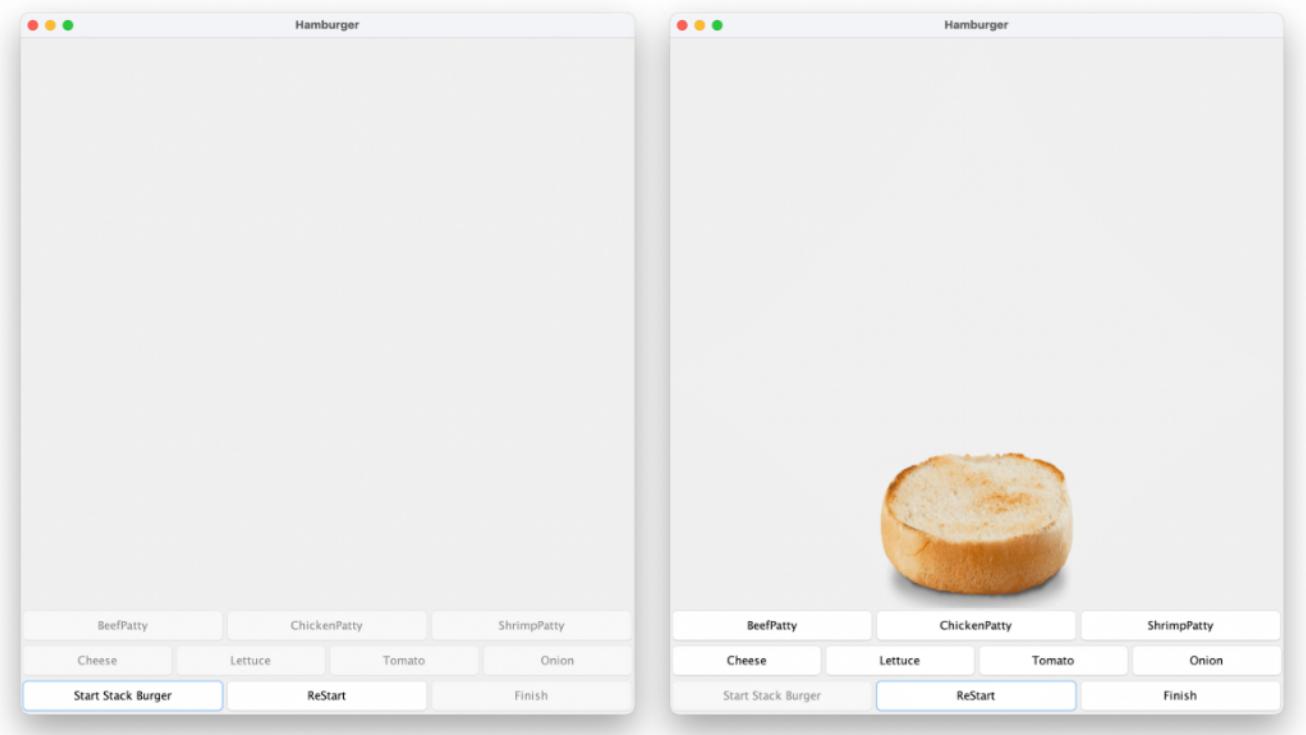
데코레이터 패턴은 기본 기능에 추가할 수 있는 기능의 종류가 많은 경웨 각 추가 기능을 Decorator 클래스로 정의한 후 필요한 Decorator 객체를 조합함으로써 추가 기능의 조합을 설계하는 방식이다. 예를 들어, 기본기능에 4가지 추가 기능이 있다고 생각해보면, 추가 기능들의 조합은 15가지나 돼서 15개의 클래스를 생성해줘야하지만, 데코레이터 패턴을 이용하면 4개의 클래스를 가지고 추가 기능의 조합을 동적으로 생성할 수 있다.



MainFrame의 JPanel의 구조는 다음과 같이 설계했다.



DisplayPanel은 추가하는 토핑들의 이미지가 들어가는 자리이고, ButtonPanel1,2,3을 GridLayout을 통해 각각의 패널에 버튼을 삽입해 주었다. 각각의 버튼이 삽입된 패널들을 ButtonPanel에 하나로 묶어서 담아주었다.
마지막으로, JFrame에는 BorderLayout을 이용하여 BorderLayout.NORTH에는 DisplayPanel을 BorderLayout.CENTER에는 ButtonPanel을 삽입시켜 각각의 패널들의 배치를 완료하였다.



Start Stack Burger를 누르게되면, MainFrame 클래스에 초기값으로 this.hamburger 저장되어있던 EmptyHamburger가 새로운 BottomBun으로 할당해주고 paintComponet 메소드를 통해 DisplayPanel에 해당 객체의 그림을 그려주고, this.hamburger.revalidate( )를 통해 변경사항을 적용해준다.

Topping 들을 추가하게되면, 현재 생성되어있는 this.hamburger 객체를 Topping의 객체의 매개변수로 넣어줘 Decorator 패턴을구현하였다. getDescription과 cost 메소드를 통해서 현재 버거 토핑의 조합과 현재까지의 버거의 가격을 출력하도록 구현했다.

```
// 각각의 버튼에 대한 구현
if(e.getSource() == buttons[0]){ // BeefPatty
    // 토핑을 더해준다. ex) BottomBun with BeefPatty를 만든다.
    this.hamburger = new BeefPatty(this.hamburger); // 데코레이터 패턴 구현을 위해 현재 this
    hamList.add(this.hamburger); // ArrayList에 추가해준다.
    System.out.println(this.hamburger.getDescription()+" ,"+this.hamburger.cost());
}else if(e.getSource() == buttons[1]){ // ChickenPatty
    this.hamburger = new ChickenPatty(this.hamburger);
    hamList.add(this.hamburger);
    System.out.println(this.hamburger.getDescription()+" ,"+this.hamburger.cost());
}else if(e.getSource() == buttons[2]){ // ShrimpPatty
    this.hamburger = new ShrimpPatty(this.hamburger);
    hamList.add(this.hamburger);
    System.out.println(this.hamburger.getDescription()+" ,"+this.hamburger.cost());
```

이렇게 각 버튼들을 구현한 후 마지막에는 공통으로 객체의 paintComponent 메소드를 실행시켜 각 버튼들에 해당하는 그림이미지를 삽입하게 구현했다.

```
this.hamburger.paintComponent(getGraphics()); // 그림을 삽입한다.
this.hamburger.revalidate(); // 새로고침
this.hamburger.repaint();
```

[추가코드 1]

```
public class MainFrame extends JFrame implements ActionListener {
    public static ArrayList<Hamburger> hamList = new ArrayList<>(); // [추가코드] Hamburger 객체를 저장하는 ArrayList
    JButton[] buttons = new JButton[10]; // JButton
    Hamburger hamburger = new EmptyHamburger(); // default
    JPanel displayPanel;

    public MainFrame(){
```

내가 추가한 코드는 ArrayList<Hamburger>이다. 이 객체는 현재 MyFrame에서 몇개의 햄버거가 얼마나 쌓여있는지 확인하기 위해 만든 객체이다. static으로 설정한 이유는 객체의 paintComponent 메소드를 호출해서 이미지를 그릴때 토핑을 몇층에 쌓여있을지 계산이 필요한데 햄버거가 얼마나 쌓여있는지 확인할 수 있는 ArrayList를 통해 쉽게 구하기 위해서 ArrayList를 static으로 구현해줬다.

```
// 이미지를 그린다.
g2.drawImage(img, x: 240, y: 500-50*(MainFrame.hamList.size()-1), width: 220, height: 200, observer: this);
```

위 그림과 같이 MainFrame으로 접근 후 객체의 size( ) 메소드를 통해 높이를 계산해줬다.

[추가코드 2]

```
buttons[9] = new JButton( text: "ReStart"); // [추가코드]
buttons[9].addActionListener( l: this); // [추가코드]
```

ReStart 버튼을 추가로 만들어주었다. 이 ReStart 버튼을 누르면 토핑을 쌓던 중간에도 처음부터 토핑쌓기가 가능하고, TopBun을 덮은후에도 이 버튼을 클릭하면 처음부터 햄버거를 쌓을 수 있다.

```
}else if(e.getSource() == buttons[9]){ // [추가코드]
    this.displayPanel.repaint(); // displayPanel을 깨끗하게 지운다.
    // Start Stack Burger 버튼을 활성화시키고, 나머지 버튼들은 비활성화
    for(int i=0;i<9;i++){
        buttons[i].setEnabled(false);
    }
    buttons[7].setEnabled(true);
}
```

이 버튼을 클릭하게 되면, displayPanel을 깨끗하게 지운뒤, StartStackBurger를 활성화시켜준다.

[실행 결과]