2018 여름 JAVA 스터디

Round 2: Graphic Editor

6. 21 – 6.25

21700581 이지현

**I. User’s Manuel**

**i ) 전체User Interface**



화면 상단에는 그리기 기능을 도와주는 도구들이 위치하고, 그 밑의 여백 캔버스 위에 그림을 그릴 수 있습니다.

**ii ) 그리기 모드**

****

**(1) Line**

: 마우스 드래그를 시작한 지점부터 놓는 지점까지 직선이 그려집니다.

**(2) Rect**

: 마우스 드래그를 시작한 지점을 왼쪽 위 꼭지점, 놓는 지점을 오른쪽 밑 꼭지점으로 하는 직사각형이 그려집니다.

**(3) Circle**

: 마우스 드래그를 시작한 위치부터 놓는 위치까지 타원이 그려집니다.

**(4) Sketch**

: 마우스 드래그 한 흔적을 따라 곡선이 그려집니다.

**(5) Spray**

: 마우스 드래그 한 흔적을 따라 스프레이가 뿌려집니다.

**(6) Pattern**

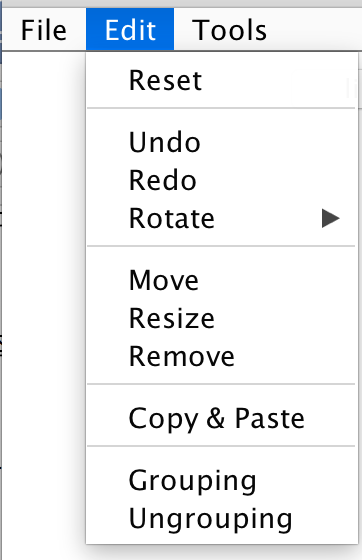
: 우선, 현재 캔버스에 그려져 있는 객체를 하나 클릭하여 선택합니다.

객체가 선택되면, 마우스 드래그 한 흔적을 따라 객체가 패턴화 됩니다.

**(7) Eraser**

: 마우스 드래그 한 흔적을 따라 캔버스를 지웁니다.

**iii ) 메뉴 바 – “Edit”**



**(1) Reset**

: 캔버스를 초기화 합니다.

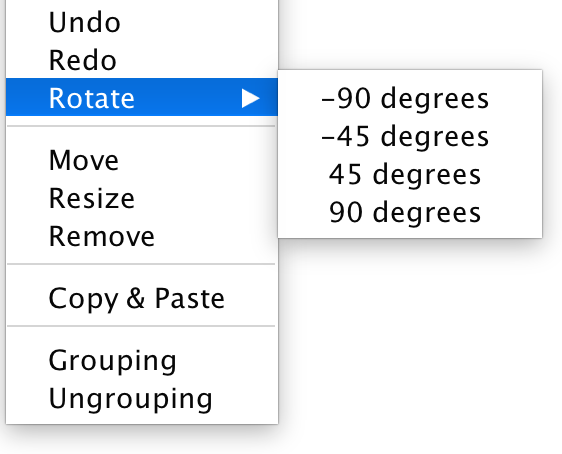
**(2) Undo**

: 가장 최근에 한 동작을 취소하고, 그 전의 캔버스 상태로 되돌립니다.

**(3) Redo**

: (2) Undo 동작을 취소하고, 원래의 캔버스 상태로 되돌립니다.

**(4) Rotate**

****

: 캔버스 위에 그려져 있는 객체를 선택하여 지정한 각도만큼 회전시킵니다.

**(5) Move**

**:** 캔버스 위에 그려져 있는 객체를 드래그하여 움직입니다.

**(6) Resize**

: 캔버스 위에 그려져 있는 객체를 드래그하여 사이즈를 조정합니다.

**(7) Remove**

: 캔버스 위에 그려져 있는 객체를 클릭하여 삭제합니다.

**(8) Copy & Paste**

: 캔버스 위에 그려져 있는 객체를 클릭하여 복사한 후,

다시 클릭하면 선택된 객체가 복사됩니다.

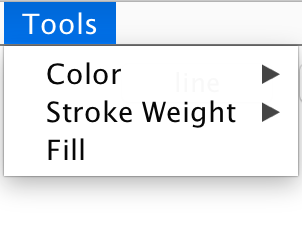
**(9) Grouping**

: 캔버스 위에 그려져 있는 객체들을 클릭하여 그룹으로 지정합니다.

**(10) Ungrouping**

: 캔버스 위에 그려져 있는 그룹을 클릭하여 그룹을 해체합니다.

**iv ) 메뉴 바 – “Tools”**



**(1) Color**

: 페인트 색깔을 조정합니다.

**(2) Stroke Weight**

: 선의 굵기나 스프레이의 두께를 조정합니다.

**(3) Fill**

: Fill 옵션이 선택되어 있으면, 닫혀 있는 객체 (원, 사각형 등) 가 현재 색상으로 채워져서 그려집니다.

**II. Technical Reference**

**1) Class Description**

**(1) Main**

: 본 프로그램의 메인 함수입니다. MainFrame의 객체를 생성하여 줍니다.

**(2) MainFrame**

: JFrame을 상속하며, 프로그램의 메인이 되는 프레임 객체의 클래스입니다.

메뉴 바, 그리기 모드 버튼 등이 프레임에 추가되어 있습니다.

**(3) Drawer**

: JPanel을 상속하며**,** MainFrame 위에 추가 될 패널입니다. 이미지 에디터에서 거의 모든 그리기 기능을 담당합니다.

**(4) DrawingMenuPanel**

: JPanel을 상속하며**,** MainFrame 위에 추가 될 패널입니다. UI 상단의 그리기 모드 버튼 (JButton) 들을 포함하고 있습니다.

**(5) Sketch**

: 팔레트 위에 그려질 “스케치”들을 객체화 하기 위하여 만든 클래스입니다. Instance variable로 Line2D들의 벡터를 가지고 있습니다. 자체적으로 *Draw* (스케치를 그리는 메소드), *isContaining* (입력받은 좌표가 스케치의 boundary에 포함되어 있는지 체크 후 true / false를 리턴하는 메소드), *move* (좌표 변화량을 받아 객체의 위치를 움직이는 메소드 등을 가지고 있습니다.

**(6) Eraser**

: 팔레트 위에 그려질 “지우개” 흔적들을 객체화 하기 위하여 만든 클래스입니다. Instance variable로 Line2D들의 벡터를 가지고 있습니다. *Draw* (스케치를 그리는 메소드), *isContaining* (입력받은 좌표가 스케치의 boundary에 포함되어 있는지 체크 후 true / false를 리턴하는 메소드), *move* (좌표 변화량을 받아 객체의 위치를 움직이는 메소드) 등을 가지고 있습니다. *Sketch* 클래스와 거의 동일하나, 색깔만 흰색으로 고정되어 있습니다.

**(6) Spray**

: 팔레트 위에 그려질 “스프레이” 들을 객체화 하기 위하여 만든 클래스입니다. Instance variable로 Ellipse2D들의 벡터를 가지고 있습니다. *Draw* (스케치를 그리는 메소드), *isContaining* (입력받은 좌표가 스케치의 boundary에 포함되어 있는지 체크 후 true / false를 리턴하는 메소드), *move* (좌표 변화량을 받아 객체의 위치를 움직이는 메소드) 등을 가지고 있습니다.

**(7) Pattern**

: 팔레트 위에 그려질 “패턴” 들을 객체화 하기 위하여 만든 클래스입니다. Instance variable로 Object들의 벡터를 가지고 있습니다. *Draw* (스케치를 그리는 메소드), *isContaining* (입력받은 좌표가 스케치의 boundary에 포함되어 있는지 체크 후 true / false를 리턴하는 메소드), *move* (좌표 변화량을 받아 객체의 위치를 움직이는 메소드) 등을 가지고 있습니다.

**(8) Group**

: Grouping 모드로 그룹 지어질 객체들을 ‘하나’의 객체로 만들기 위하여 만든 클래스입니다. Instance variable로 Object들의 벡터를 가지고 있습니다. *Draw* (스케치를 그리는 메소드), *isContaining* (입력받은 좌표가 스케치의 boundary에 포함되어 있는지 체크 후 true / false를 리턴하는 메소드), *move* (좌표 변화량을 받아 객체의 위치를 움직이는 메소드), *ungroup* (객체들을 다시 ungrouping 하는 메소드) 등을 가지고 있습니다.

**(9) Edit\_Copy, Edit\_Grouping, Edit\_Move, Edit\_Redo, Edit\_Reset, Edit\_Resize, Edit\_Rotate, Edit\_Resize, Edit\_Rotate, Edit\_Undo, Edit\_Ungrouping**

**:** 메뉴 바의 “Edit” 카테고리에 포함된 기능들을 지원하는 클래스들입니다. JMenuItem 을 상속하며, 각자 맡은 기능 구현에 필요한 메소드를 포함하고 있습니다.

**(10) Menu\_Color, Menu\_Fill, Menu\_Thickness**

: 메뉴 바의 “Tools” 카테고리에 있는 기능들을 지원하는 클래스들입니다. JMenuItem 을 상속하며, 각자 맡은 기능 구현에 필요한 메소드를 포함하고 있습니다.

**2) General Flow**

**(1)** *Main* 클래스의 메인 함수에서 *MainFrame* 클래스의 객체를 생성합니다. *MainFrame* 객체에는 *Drawing* 패널과 *DrawingMenuPanel* 패널이 추가되어 있습니다.

**(2)** 유저가 DrawingMenuPanel 의 버튼들 (line, rect, circle, etc) 을 클릭하면, *Drawing* 패널의 instance variable인 *whichShape* 의 값이 변합니다.

(*int* *whichShape* : 현재 어떤 모드에 있는지 알려주는 변수)

**(3)** 유저의 마우스 액션을 받으면,*whichShape*의 값에 대응하는 도형을 그립니다. 다 그려진 도형은 *Drawing* 패널의 instance variable인*object\_vector* 에 추가합니다.

(*Vector<Object> object\_vector*: 그리기 완료된 도형들을 담고 있는 벡터)

도형에 관한 정보 – 색깔, 선의 굵기, fill 여부 – 들도 벡터에 추가해 둡니다.

(*Vector<Color> color\_vector*: 도형의 색깔을 담고 있는 벡터)

(*Vector<Integer> thickness\_vector*: 도형의 선 굵기를 담고 있는 벡터)

(*Vector<Integer> fill\_vector*: 도형의 fill 여부를 담고 있는 벡터; 값이 -1이면 no fill, 1이면 fill)

**(4)** 유저가 새로운 도형을 그리기 시작한다면, public void *paintComponent (Graphics g)* 메소드 안에 이전 도형들을 모두 그려주는 메소드 *public void paintPreviousThings (Graphics2D g2)* 도 같이 호출합니다. 따라서, 유저가 새로운 도형을 그리는 동안에도 이전에 그렸던 도형이 잔상이 남아있을 수 있습니다.

(*public void* *paintPreviousThings (Graphics2D g2)*: *object\_vector, color\_vector, thickness\_vector, fill\_vector* 에 저장된 값을 이용하여 유저가 이전에 그렸던 도형들을 모두 그려주는 메소드)

**3) Algorithms**

**(1) Reset**

: 지금까지 그려진 도형에 대한 정보를 저장하고 있는 벡터를 모두 초기화합니다.

**(2) Undo / Redo**

: Undo - 지금까지 그려진 도형에 대한 정보를 저장하고 있는 벡터들 *(object\_vector, color\_vector, thickness\_vector, fill\_vector)* 안에서 가장 최근 추가된element를 제거하고, *temp\_copy* 라는 임시 벡터에 저장합니다.

Redo *–* temp\_copy 임시 벡터에 저장된 도형 정보들을 불러와 다시 *object\_vector, color\_vector, thickness\_vector, fill\_vector* 에 저장합니다.

**(3) Move**

: 우선, 마우스 입력이 특정 객체의 boundary 내에 있는지 확인합니다. 맞다면, 마우스 드래그 시 마우스 포인터가 움직인 x, y 변화량을 계산한 후, *setRect, setFrame* 등의 메소드를 이용하여 해당 객체의 위치를 다시 지정하여 줍니다.

**(4) Resize**

: 우선, 마우스 입력이 특정 객체의 boundary 내에 있는지 확인합니다. 맞다면, 마우스 드래그 시 마우스 포인터가 움직인 x, y 변화량을 계산한 후, *setRect, setFrame* 등의 메소드를 이용하여 해당 객체의 사이즈를 다시 지정하여 줍니다.

**(5) Copy & Paste**

:우선, 마우스 입력이 특정 객체의 boundary 내에 있는지 확인해주고, 객체가 선택되었다면*temp\_copy* 라는 임시 벡터로 deep copy 합니다. 그 후 paste 모드로 전환되는데, 유저가 클릭하는 마우스 좌표로 복사했던 객체를 move 한 뒤, *object\_vector* 에 저장합니다.

**(6) Grouping / Ungrouping**

: Grouping -마우스 입력이 특정 객체의 boundary 내에 있는지 확인해주고, 새 Group 클래스의 객체를 만들어, 선택된 객체들을 Group 객체로 *add* 하여 줍니다

(*add* 메소드: 파라미터로 받은 객체를 *Group* 객체의 instance variable로 넣어주고, 원래의 object\_vector에서는 제거함).

Ungrouping – Grouping 에서 했던 것을 반대로 행하여 줍니다 (*Group* 객체의 instance variable들을 원래의 object\_vector에 저장해주고, *Group* 객체를 제거함).

**4) Suggestions**

**(1) 중복되는 코드들**

본 프로그램의 코드 중에는 중복되는 부분들이 많습니다. 예를 들어, 반복문을 이용하여 벡터 안에 저장된 Object들을 모두 그려주는 작업이나, 마우스 클릭을 받아 *object\_vector* 중 어떠한 객체가 선택되었는지 알려주는 구문들이 반복적으로 등장하는데, 이를 메소드로 만들어 만들어 중복되는 부분을 줄이면 좋았을 것이라는 아쉬움이 남습니다.

**(2) Generics**

코드 중, Object 변수에 저장된 객체의 인스턴스를 일일히 따져가며, 그에 적합한 작업을 수행하는 구문이 많습니다. 이를 Generics 개념을 적용한 메소드로 구현하면 코드의 간결성, 가독성이 높아질 것 같습니다.

**(3) 코드 가독성**

본 코드의 가독성에 대한 아쉬움이 많이 남습니다. 예를 들어, *Drawing* 클래스의 instance variable인 *whichShape*의 타입을 String이나 Enum 타입으로 만들어, 변수의 값이 어떠한 그리기 모드를 나타내는지 직접적으로 나타냈으면 좋았을 것입니다.

**(4) User Interface**

UI를 시각적으로 더 개선했으면 좋겠습니다. 또한, 키보드 입력이나 마우스 우클릭 등을 이용하여 유저 인풋을 다양하게 받는다면 보다 편리한 프로그램이 되었을 것입니다.