실습 1:

-실습코드

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

-결과출력

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

-코드관련 설명

#Pencil , brush 클래스와 같은 형태로 Crayon 클래스를 만듭니다.

#DrawingFrame 클래스 안에 drawingCrayon 메소드를 작성하여 Crayon 객체 c 를 받아 draw(); 메소드가 작동하게 합니다.

# Crayon 객체 c1,c2을 만듭니다. C1은 기본 생성자를 호출하고, c2는 7,”yellow”를 파라미터로 합니다.

#Arraylist<Crayon> crayonList를 만든 후 , crayonList에 Crayon 객체 c1,c2를 추가합니다.

#이후 for 문을 통해 DrawingFrame 객체 f의 메소드 drawingCrayon을 실행합니다.

실습 2:

1. 실습1번 코드의 문제점:

실습1번 코드에는 Brush와 Pencil, Crayon이라는 별도의 리스트가 있어 drawingBrush()와 drawingPencil() drawingCrayon() 메서드에서 각각 호출되고 있습니다. 이처럼 도구의 종류마다 별도의 메서드와 리스트를 사용하는 것은 다형성을 충분히 활용하지 않는 비효율적인 구조입니다.

또한 Open-Closed Principle에 위배됩니다.

현재 구조에서는 새로운 도구 타입(예: Crayon 클래스)이 추가될 때마다 DrawingFrame에 새로운 메서드나 조건을 추가할 가능성이 큽니다. 이는 코드 수정이 필요하다는 의미로, OCP에 위배됩니다

마지막으로 Dependency Inversion Principle에 위배됩니다.

DrawingFrame 클래스는 특정 구현(Brush, Pencil)에 직접 의존하고 있습니다. DIP에 따르면, DrawingFrame은 Tool이라는 추상화에 의존해야 하며, 구체적인 구현에는 의존하지 않아야 합니다.

-실습 코드

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

-결과 출력

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

-코드관련 설명

#추상클래스 Tool을 만듭니다. 추상메소드 draw() 를 만들고,

#Tool 클래스를 pencil, brush, Crayon 클래스에 상속시킵니다. 따라서 추상클래스 Tool의 추상메소드 draw()를 , pencil, brush, Crayon 클래스에서 다 구현할 수 있습니다.

# 이후 DrawingFrame 클래스 안에

public void drawing(Tool tool) {

tool.draw(); }

메소드를 만들고, 다형성을 구현합니다.

#ArrayList<Tool> toolList를 만들고 그곳에 객체 b1,b2,p1,p2,c1,c2를 저장합니다.

#for 문을 통해 toollist 속의 모든 객체들을 tool 객체로 업캐스팅 하고, (ex, Tool tool=Pencil p1) 이 때의 tool을 파라미터로 객체 f의 메소드 drawing을 작동시킵니다. 그러면 각각의 tool 객체마다 draw() 메소드가 발동되는데, 이때 메소드 오버라이딩을 통해 tool 객체의 하위 클래스인 Brush, Pencil, Crayon의 draw()메소드가 발동됩니다.

실습 3:

-실습 코드:

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명-결과 출력:

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

-코드 설명:

#Tool을 인터페이스로 정의하여 draw 메서드를 구현하도록 했고,

#이후 Erase 인터페이스를 새로 정의하고 erase 메서드를 추가했습니다.

#Tool과 Erase를 동시에 구현하는 MechanicalPencil 클래스를 작성하였고

(class MechanicalPencil implements Erase, Tool)

setColor(),setSize() 메소드, 기본생성자, 생성자들의 형태는 crayon, pencil, brush 클래스와 같은형태로 만들었습니다.

또한 이 클래스는 draw와 erase 메서드를 모두 구현합니다.

# drawing 메서드를 통해 Tool 인터페이스를 사용하는 객체를 그릴 수 있고,

MechanicalPencil m1,MechanicalPencil m2 객체를 생성합니다. 이후 ArrayList<Tool> toolList에 toolList.add(m1); toolList.add(m2); 메소드를 통해 m1,m2 객체를 넣은 후

For 문을 통해 각 객체별로 drawing 메소드를 적용해 메소드 오버라이딩을 통해 draw() 메소드를 적용합니다.