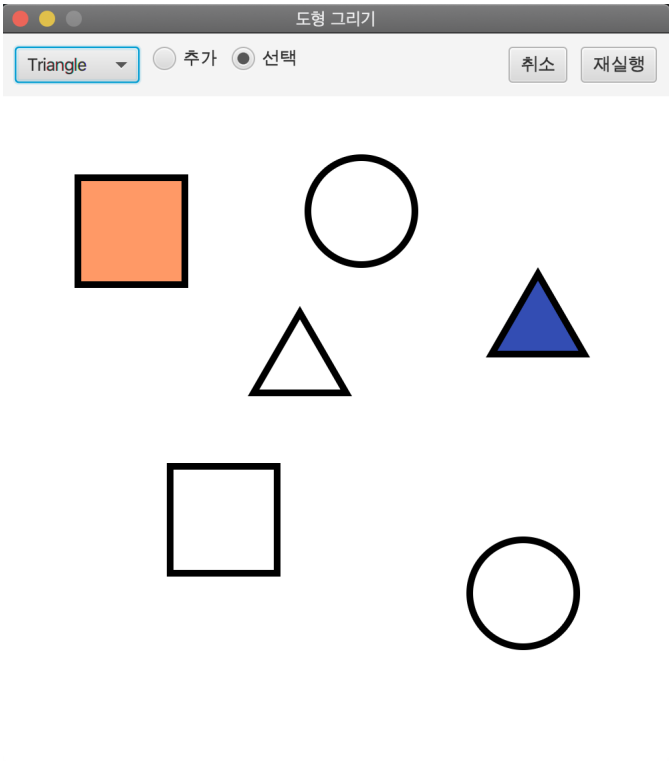
객체지향개발론및실습

Laboratory 6. Command 패턴

1. 목적

- 요청을 객체로 캡슐화해주며, 파라미터를 통해 다양한 요청을 처리할 수 있도록 해주는 명령 패턴을 실습해 본다.
- 이 실습에서 자바는 JavaFX, C++는 Qt6, 파이썬은 PyQt6 라이브러리를 이용하여 GUI 기반 실습을 수행한다.

2. 구현할 프로그램의 주 화면



- 이 응용은 3종류(정사각형, 원, 정삼각형)의 도형을 그릴 수 있고, 그린 도형을 선택하여 채우기 색을 변경할 수 있고, 그린 도형을 선택하여 지울 수 있다.
- 제시한 3가지 작업에 대해 수행한 것을 취소할 수 있으며, 취소할 경우 재실행할 수 있다.

3. 실습 1. 명령 패턴 없이 구현하기

3.1 자바

- JavaFX에서는 이 응용을 개발하기 위해 **Canvas**에 각종 그리기 메소드를 이용하여 그릴 수 있고, **Pane**에 도형 객체를 추가하여 그릴 수 있다.
- 이 실습에서는 **Pane**에 그리고자 하면 도형에 따라 **Rectangle**, **Circle**, **Polygon** 객체를 생성하여 추가하는 방법으로 도형을 그린다.

3.2 C++와 파이썬

- 자바와 유사하게 **QGraphicsRectItem**, **QGraphicsEllipseItem**, **QGraphicsPolygonItem** 객체를 생성하여 **QGraphicsScene**에 추가하여 도형을 그린다.

3.3 구현 요구사항

- 수행한 것을 취소하는 부분과 취소한 것이 있을 때 재실행하는 부분을 제외한 동작하는 소스를 제공한다.
- 주어진 소스를 수정하여 취소와 재실행이 동작하도록 구현한다. 이때 명령 패턴을 활용하지 않고 구현한다.
- 명령 패턴을 이용하지 않더라도 지금까지 수행한 작업의 순서를 기억해야 하고, 이것을 기억하기 위해 스택의 활용은 필요하다. 반드시 스택을 사용해야 하는 것은 아니다. 자유롭게 본인의 생각대로 취소와 재실행이 올바르게 동작하기만 하면 된다.

4. 실습 2. 명령 패턴을 이용하여 구현하기

- 명령 패턴을 이용하기 위해서는 **Command interface**를 정의하고 이를 이용하여 각 작업을 수행하는 명령 클래스를 정의해야 한다.

```
public interface Command {
    void execute();
    void undo();
}
```

파이썬은 다음과 같은 클래스를 정의하여 사용한다.

```
class Command(ABC):
    @abstractmethod
    def execute(self):
        pass

    @abstractmethod
    def undo(self):
        pass
```

C++는 다음과 같은 클래스를 정의하여 사용한다.

```
class Command{
public:
    virtual ~Command() = default;
    virtual void execute() = 0;
    virtual void undo() = 0;
};
```

- 총 5개(3종류의 도형 그리기, 도형 삭제하기, 도형 색 변경하기)의 명령 클래스를 만들 수 있고, 3개의 명령 클래스(도형 그리기, 도형 삭제하기, 도형 색 변경하기)를 만들어 구현할 수 있다.

- 취소와 재실행은 두 개의 스택을 만들고 명령 객체를 유지하여 구현한다.
- 자바
  - 제공한 프로젝트의 **DrawShapeApp** 클래스의 **drawShape** 메소드에서 각종 도형 객체를 생성하여 도형을 추가하는 방법으로 응용을 구현하였다.
  - 이 실습에서는 **drawShape** 메소드에서 필요한 명령 객체를 생성한 후에 이 명령 객체의 **execute** 메소드를 실행하여 도형을 그리도록 수정해야 한다.
- C++와 파이썬
  - 제공한 프로젝트의 **Canvas** 클래스의 **drawShape** 메소드에서 각종 도형 객체를 생성하여 주는 메소드를 호출하여 도형을 화면에 추가하는 방법으로 응용을 구현하였다.
  - 이 실습에서는 **drawShape** 메소드에서 필요한 명령 객체를 생성한 후에 이 명령 객체의 **execute** 메소드를 실행하여 도형을 그리도록 수정해야 한다.

5. 실습 3. 명령 관리자를 이용하여 구현하기

- 이전 실습에서는 명령 객체를 생성한 후에 직접 실행하였고, 취소 및 재실행을 위한 스택을 관리하는 별도 객체가 없었다.
- 이 실습에서는 명령 객체를 관리하는 다음과 같은 명령 관리자를 구현하여 명령 객체의 실행과 취소 및 재실행을 위한 스택을 이 객체가 관리하도록 수정한다.

```
public class CommandManager{
    private Stack<Command> undoStack = new Stack<>();
    private Stack<Command> redoStack = new Stack<>();
    public void execute(Command command){}
    public boolean undo(){}
    public boolean redo(){}
}
```

파이썬에서 명령 관리자의 모습은 다음과 같다.

```
class CommandManager:
    def __init__(self):
        self.undoList = deque()
        self.redoList = deque()

    def execute(self, command: Command):
        pass

    def undo(self) -> bool:
        pass

    def redo(self) -> bool:
        pass
```

C++의 명령 관리자는 다음과 같다.

```
class CommandManager{
    QList<Command*> undoList;
    QList<Command*> redoList;
public:
    CommandManager() = default;
    ~CommandManager();
    void execute(Command* command);
    bool undo();
    bool redo();
};
```

여기서 **undo**와 **redo**의 반환값은 버튼의 disable 여부를 처리하기 위해 필요하여, 취소 또는 재실행을 요청하였을 때 할 것이 없으면 **false**를 반환하고, 요청을 정상적으로 수행하였으면 **true**를 반환하도록 구현한다.

- 자바의 경우 **execute**는 가변 인자를 사용하여 하나가 아니라 여러 개를 실행할 수 있도록 만들 수 있다.

6. Qt6

- Qt6로 응용을 개발할 때 C++ 표준 라이브러리에 있는 각종 자료구조와 클래스를 활용할 수 있다. 하지만 그 대신에 Qt에서 제공하는 자료구조를 사용하는 것이 Qt의 다른 메소드와 연동하는 측면에서 더 편리한 측면이 있다. 특히, Qt의 메소드에서 문자열을 처리할 때 **QString**을 사용하므로 **std::string** 대신에 **QString**을 사용하는 것이 더 효과적이다.

7. 차이점

- 이 실습에서는 같은 응용을 명령 패턴을 이용하지 않는 버전, 이용하는 버전, 명령 관리자를 이용하는 방법 총 3가지 버전으로 구현해 보았다. 이들의 차이점은? 특히, 명령 패턴을 이용한 이점이 있는지 생각해 보자.