

자료구조 실습 보고서

[제01주] 마방진

2021년 3월 14일

201702039 오명주

1. 프로그램 설명서

(1) 프로그램의 전체 설계 구조

➔ MVC (Model – View – Controller) 구조

Model : 프로그램이 “무엇”을 할 것인지 정의. 사용자의 요청에 맞는 알고리즘을 처리하고 DB와 상호작용하여 결과물을 산출하고 Controller에게 전달.

View : 화면에 무엇인가를 “보여주기 위한” 역할. 최종 사용자에게 “무엇”을 화면으로 보여줌.

Controller : 모델이 “어떻게” 처리할 지 알려주는 역할. 사용자로부터 입력을 받고 중개인 역할. Model과 View는 서로 직접 주고받을 수 없음. Controller을 통해 이야기함.

➔ 마방진 프로그램에서의 각 클래스 별 MVC 구조 역할

Model :

- MagicSquare : 입력 받은 차수를 통해 마방진을 계산한다.
- Board : 계산된 마방진 판을 만들고 숫자를 넣는다.
- CellLocation : 마방진의 각 Cell 좌표를 구성한다.
- OrderValidity : 입력 받은 차수의 유효성을 검사하는 enum 이다.

View :

- AppView : 프로그램의 입/출력을 담당한다.

Controller :

- AppController : AppView를 통해 차수를 입력받고 Model 의 클래스에 전달하고 결과물인 마방진 판을 AppView를 통해 출력한다.

(2) 함수 설명서

➔ 주요 알고리즘

```
for (int cellValue = 2; cellValue <= lastValue; cellValue++) {  
    // 단계 1: <현재 위치>로부터 <다음 위치>인 "오른쪽 위" 위치를 계산한다  
    // 만약 "오른쪽 위" 위치가 주어진 행과 열을 벗어난다면 다시 조정한다  
    if (currentLoc.row() - 1 < 0 && currentLoc.col() + 1 >= anOrder) {  
        nextLoc.setRow(currentLoc.row() - 1 + anOrder);  
        nextLoc.setCol(currentLoc.col() + 1 - anOrder);  
    }  
    // 만약 "오른쪽 위" 위치가 주어진 행을 벗어난다면 가장 아래쪽으로 위치를 다시 조정한다  
    else if (currentLoc.row() - 1 < 0) {  
        nextLoc.setRow(currentLoc.row() - 1 + anOrder);  
        nextLoc.setCol(currentLoc.col() + 1);  
    }  
    // 만약 "오른쪽 위" 위치가 주어진 열을 벗어난다면 가장 좌측으로 위치를 다시 조정한다.  
    else if (currentLoc.col() + 1 >= anOrder) {  
        nextLoc.setRow(currentLoc.row() - 1);  
        nextLoc.setCol(currentLoc.col() + 1 - anOrder);  
    }  
    // 만약 "오른쪽 위" 위치가 주어진 행과 열을 벗어나지 않는다면 좌표를 설정한다  
    else {  
        nextLoc.setRow(currentLoc.row() - 1);  
        nextLoc.setCol(currentLoc.col() + 1);  
    }  
    // 단계 2: <다음 위치>가 채워져 있으면  
    // <다음 위치>를 <현재 위치>의 바로 한 줄 아래 칸 위치로 수정한다  
    if (!board.cellIsEmpty(nextLoc)) {  
        nextLoc.setRow(currentLoc.row() + 1);  
        nextLoc.setCol(currentLoc.col());  
    }  
    // 단계 3: <다음 위치>를 새로운 현재 위치로 한다  
    currentLoc.setRow(nextLoc.row());  
    currentLoc.setCol(nextLoc.col());  
    // 단계 4: 새로운 현재 위치에 number 값을 넣는다  
    board.setCellValue(currentLoc, cellValue);  
}
```

마방진 프로그램에서 nextLoc Cell 위치 잡는 논리 : row - 1, col + 1 (현재 위치의 오른쪽 위 좌표로 설정)

- 1) 만약 nextLoc 위치가 row를 벗어난다면 row - 1에 입력 받은 차수만큼 더해준다.
- 2) 만약 nextLoc 위치가 col을 벗어난다면 col + 1에 입력 받은 차수만큼 빼준다.
- 3) 만약 nextLoc 위치가 row, col를 모두 벗어난다면 1)와 2) 모두 해준다.

→ 이 클래스(MagicSquare)의 전체적인 알고리즘은 다음과 같다.

```
public Board solve(int anOrder) {
    if (OrderValidity.validityOf(anOrder) != OrderValidity.Valid) {
        return null;
    } else {
        Board board = new Board(anOrder);
        // 차수와 함께 Board 객체 생성자를 call 하여 , Board 객체를 생성한다
        CellLocation currentLoc = new CellLocation(0, anOrder / 2);
        // 출발 위치 보드의 맨 윗줄 한 가운데를 현재의 위치로 설정
        CellLocation nextLoc = new CellLocation();
        board.setCellValue(currentLoc, 1);
        // 보드의 출발 위치 에 1 을 채운다
        int lastValue = anOrder * anOrder;
    }
}
```

Board 클래스의 판을 입력 받은 차수 anOrder*anOrder 크기만큼 생성한 후, 보드의 맨 윗줄 가운데를 시작 위치로 설정한다. 시작 위치인 currentLoc에 1을 넣은 후 2부터는 앞서 설명한 nextLoc 설정 알고리즘을 for문으로 반복한 후 nextLoc을 currentLoc으로 설정하고 board에 setCellValue 함수를 이용하여 값을 넣는다.

→ Board 클래스는 다음과 같다.

```
// 기본 생성자
public Board(int givenOrder) {
    this.setOrder(givenOrder);
    this.setCells(new int[givenOrder][givenOrder]);
    for(int row = 0; row < givenOrder; row++) {
        for(int col = 0; col < givenOrder; col++) {
            this.setCellValue(row, col, Board.EMPTY_CELL);
        }
    }
}

// 공개 함수 (public methods)
public boolean cellIsEmpty(CellLocation location) { // "Cell is empty?"
    // 주어진 위치의 cell이 비어 있는지 여부를 알려준다
    // 비어 있으면 true, 아니면 false를 얻는다
    return (this.cellValue(location) == EMPTY_CELL);
}
```

마방진 보드 생성자가 있으며 입력 받은 차수 크기의 보드를 생성한다. 해당 Cell이 비었는지 안 비었는지 확인하는 cellIsEmpty 함수도 존재한다. 비었으면 true를, 채워져 있으면 false를 반환한다.

```

public int cellValue(CellLocation location) {
    // 주어진 location의 cell값을 얻는다
    return this.cells()[location.row()][location.col()];
}

public void setCellValue(CellLocation location, int newCellValue) {
    // 주어진 location의 cell에 주어진 value를 넣는다
    this.cells()[location.row()][location.col()] = newCellValue;
}

private void setCellValue(int row, int col, int newCellValue) {
    // 이 method는 class 내부에서만 사용한다
    // 주어진 위치 (row, col)의 cell에 주어진 값 value를 넣는다
    this.cells()[row][col] = newCellValue;
}

```

Cell에 값을 넣는 getter/setter 이 존재한다. 주어진 Location에 Cell값을 넣는 함수와 해당 Location에 주어진 value를 넣는 함수가 있다.

➔ 마방진에서 Cell 좌표를 구성하는 **CellLocation** 클래스는 다음과 같다.

```

// 기본 생성자 : Cell 좌표가 주어지지 않는다
public CellLocation() {
    // Cell좌표가 주어지지 않으면 (-1, -1)로 설정하기로 한다
    this.setRow(UNDEFINED_INDEX);
    this.setCol(UNDEFINED_INDEX);
}

// Cell 좌표가 주어지는 생성자
public CellLocation(int givenRow, int givenCol) {
    this.setRow(givenRow);
    this.setCol(givenCol);
}

```

기본 생성자는 다음과 같다. Cell의 좌표가 주어지지 않으면 상수 UNDEFINED_INDEX (여기서는 -1로 미리 정의해 놓았다) 로 설정한다. 주어진 차수로 설정하는 생성자도 존재 한다.

```
// Getter / Setter
// 위치의 row 좌표를 입력받아 설정한다
public void setRow(int newRow) {
    this._row = newRow;
}
// 위치의 row 좌표를 반환한다
public int row() {
    return this._row;
}
// 위치의 col 좌표를 입력받아 설정한다
public void setCol(int newCol) {
    this._col = newCol;
}
// 위치의 col 좌표를 반환한다
public int col() {
    return this._col;
}
```

해당 클래스의 getter / setter 함수들이다. Row, col의 위치 좌표를 반환하고 해당 위치로 설정한다.

➔ 각 클래스와 함수마다 기능이 명확히 구분되어 있어 다소 복잡하지만 높은 생산성과 유연성을 가지고 있다.

(3) 종합 설명서

➔ 프로그램 실행 순서대로 설명해보자.

```
public class _DS01_Main_201702039_오명주 {
    public static void main(String[] args) {
        AppController appController = new AppController();
        // AppController가 실질적인 main class
        appController.run();
    }
}
```

Main 클래스에서 AppController 객체 생성 후 appController의 run 메소드를 실행한다.

```

// 공개함수
public void run() {
    AppView.outputLine("<<< 마방진 풀이를 시작합니다 >>>");
    AppView.outputLine("");
    AppView.output("? 마방진 자수를 입력하시오(음수를 입력하면 종료합니다): ");

    int currentOrder = AppView.inputOrder(); // 메시지를 내보내고 자수를 입력받음
    OrderValidity currentValidity = OrderValidity.validityOf(currentOrder);
    while (currentValidity != OrderValidity.EndOfRun) { // 자수가 음수이면 프로그램 종료
        if (currentValidity == OrderValidity.Valid) { // 자수가 유효한지 검사
            AppView.outputTitleWithOrder(currentOrder);
            Board solvedBoard = this._magicSquare.solve(currentOrder);
            // _magicSquare 객체에게 주어진 자수의 마방진을 풀도록 시킨다.
            // 결과로 마방진 판을 얻는다
            this.showBoard(solvedBoard); // 마방진을 화면에 보여준다
        } else {
            this.showOrderValidityErrorMessage(currentValidity);
        }
        AppView.outputLine("");
        AppView.output("? 마방진 자수를 입력하시오(음수를 입력하면 종료합니다): ");
        currentOrder = AppView.inputOrder(); // 다음 마방진을 위해 자수를 입력받음
        currentValidity = OrderValidity.validityOf(currentOrder);
    }
    AppView.outputLine("");
    AppView.outputLine("<<< 마방진 풀이를 종료합니다 >>>");
}
}

```

AppController의 run 메소드에서는 자수를 입력 받기 위한 출력문들과 입력 받은 자수의 유효성을 확인하는 함수를 호출한다. 이때, 입력 받을 때는 AppView 클래스의 함수를 이용한다. 만약 자수가 유효하다면 magicSquare 클래스에 전달하여 결과를 얻는다. showBoard 함수를 이용하여 결과 마방진을 출력한다. 이때, 마방진을 계산하는 알고리즘은 위 페이지를 참고한다.

➔ 이때, 입/출력을 담당하는 **AppView** 클래스는 다음과 같다.

```

// 입력 관련 함수
public static int inputOrder() {
    int inputValue = scanner.nextInt(); // scanner를 이용하여 정수를 입력받는다
    return inputValue;                 // 입력받은 정수를 return한다
}

```

입력 관련 함수에는 자수 입력 받는 inputOrder 함수가 존재한다. 자바 표준 입력 클래스 Scanner를 import하여 자수를 입력 받아 입력 받은 자수를 return 한다.


```

// 볼록한 단 함수
// 한줄을 출력하는 함수 (한줄이 띄워지지 않는다)
public static void output(String message) {
    System.out.print(message);    // 입력받은 message를 출력한다
}

// 한줄을 출력하는 함수 (한줄이 띄워진다)
public static void outputLine(String message) {
    System.out.println(message);    // 입력받은 message를 출력한다
}

public static void outputTitleWithOrder(int order) {
}

// 마방진에 줄을 나타내는 함수
public static void outputRowNumber(int number) {
    System.out.printf("[%3d] ", number);    // number를 입력받아 주어진 형식으로 출력한다
}

// 마방진 수를 채우는 함수
public static void outputCellValue(int value) {
    System.out.printf("  %3d ", value);    // value를 입력받아 마방진 주어진 위치에 수를 출력한다
}

```

Console 창의 출력문을 출력하는 함수들과 마방진을 출력할 때 행과 열을 나타내는 숫자와 각 수를 주어진 위치에 출력하는 함수들이 있다. 각각의 상황에 맞게 print문의 구조를 다르게 설정한다. "%3d"는 3칸 중 뒤에서부터 값을 출력하는 형태이다.

2. 프로그램 장단점 / 특이점 분석

➔ 장점

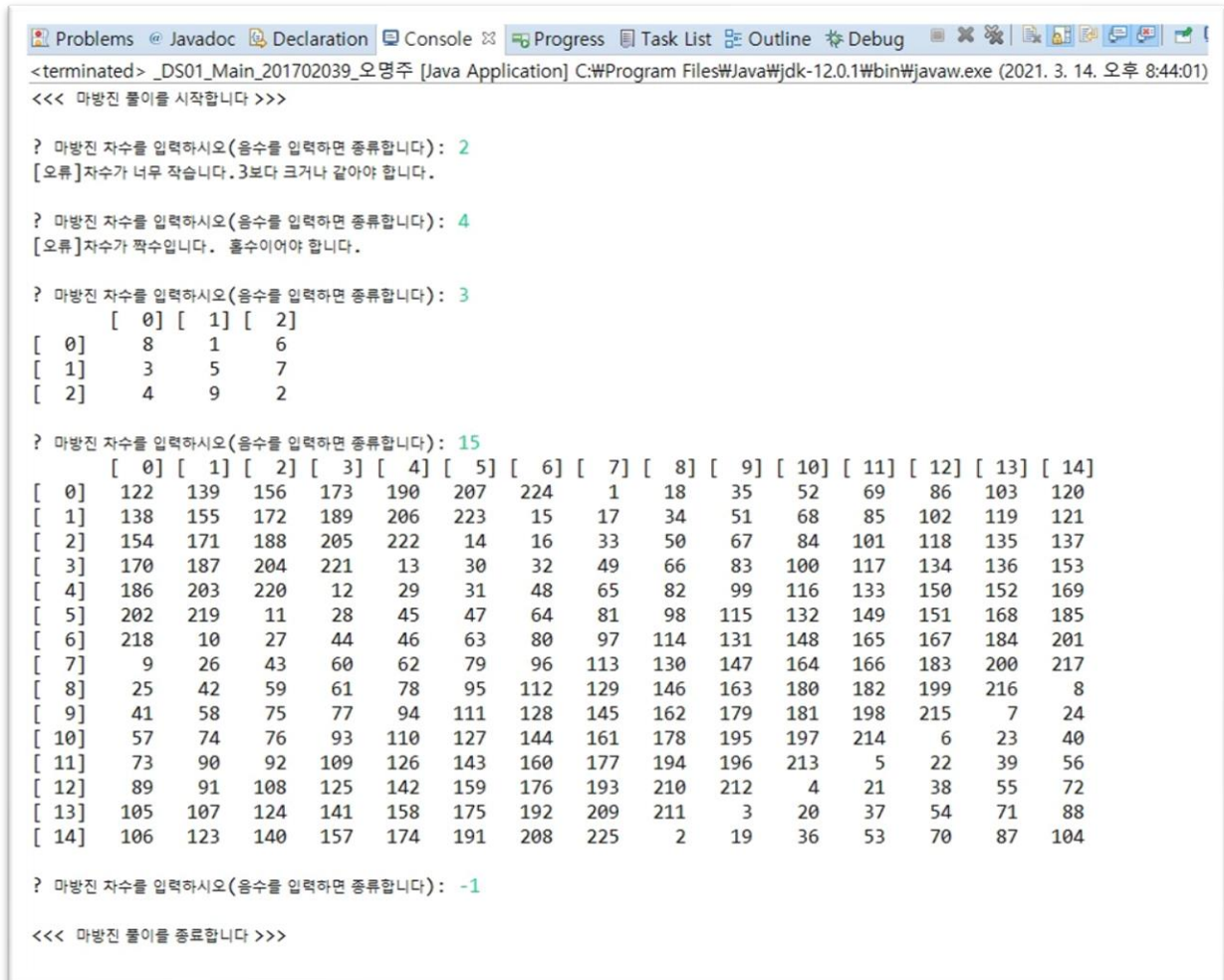
- MVC 모델을 이용하여 가독성과 생산성이 뛰어나다. 각 클래스, 함수의 역할이 분명해서 코드와 프로그램을 잘 이해할 수 있다.
- 유사한 기능을 하는 다른 프로그램에도 재사용할 수 있다. 객체 지향 프로그램의 가장 큰 장점이라고 할 수 있다.
- 수정이 편리하다. 데이터나 기능을 수정하려고 하면 해당 메소드만 수정하면 되기 때문에 편리하다.

➔ 단점

- 처음에 클래스와 함수 역할을 뚜렷하게 나누는 것이 쉽지 않다. 객체 지향 프로그램 설계할 때 시간이 오래 걸린다.
- 입/출력까지 모두 분리 하다 보니 코드양이 많아지고 시간이 오래 걸린다.

3. 실행 결과 분석

(1) 입력과 출력 (화면 capture하여 제출)



```
<terminated> _DS01_Main_201702039_오명주 [Java Application] C:\Program Files\Java\jdk-12.0.1\bin\javaw.exe (2021. 3. 14. 오후 8:44:01)
<<< 마방진 풀이를 시작합니다 >>>

? 마방진 자수를 입력하십시오(음수를 입력하면 종료합니다): 2
[오류]자수가 너무 작습니다. 3보다 크거나 같아야 합니다.

? 마방진 자수를 입력하십시오(음수를 입력하면 종료합니다): 4
[오류]자수가 짝수입니다. 홀수이어야 합니다.

? 마방진 자수를 입력하십시오(음수를 입력하면 종료합니다): 3
[ 0] [ 1] [ 2]
[ 0] 8 1 6
[ 1] 3 5 7
[ 2] 4 9 2

? 마방진 자수를 입력하십시오(음수를 입력하면 종료합니다): 15
[ 0] [ 1] [ 2] [ 3] [ 4] [ 5] [ 6] [ 7] [ 8] [ 9] [ 10] [ 11] [ 12] [ 13] [ 14]
[ 0] 122 139 156 173 190 207 224 1 18 35 52 69 86 103 120
[ 1] 138 155 172 189 206 223 15 17 34 51 68 85 102 119 121
[ 2] 154 171 188 205 222 14 16 33 50 67 84 101 118 135 137
[ 3] 170 187 204 221 13 30 32 49 66 83 100 117 134 136 153
[ 4] 186 203 220 12 29 31 48 65 82 99 116 133 150 152 169
[ 5] 202 219 11 28 45 47 64 81 98 115 132 149 151 168 185
[ 6] 218 10 27 44 46 63 80 97 114 131 148 165 167 184 201
[ 7] 9 26 43 60 62 79 96 113 130 147 164 166 183 200 217
[ 8] 25 42 59 61 78 95 112 129 146 163 180 182 199 216 8
[ 9] 41 58 75 77 94 111 128 145 162 179 181 198 215 7 24
[ 10] 57 74 76 93 110 127 144 161 178 195 197 214 6 23 40
[ 11] 73 90 92 109 126 143 160 177 194 196 213 5 22 39 56
[ 12] 89 91 108 125 142 159 176 193 210 212 4 21 38 55 72
[ 13] 105 107 124 141 158 175 192 209 211 3 20 37 54 71 88
[ 14] 106 123 140 157 174 191 208 225 2 19 36 53 70 87 104

? 마방진 자수를 입력하십시오(음수를 입력하면 종료합니다): -1
<<< 마방진 풀이를 종료합니다 >>>
```

(2) 결과 분석 (자신의 논리적 평가, 기타 느낀 점)

객체 지향 설계가 편리함은 이미 알고있었고, 가독성이나 재사용성이 높다는 것도 알고 있었으나 프로그램을 구현할 때 각각의 클래스나 함수가 어떠한 역할을 맡는지 명확하게 나누는 것이 나로서는 어려웠다. 하지만 주어진 클래스들을 분석하고 어떠한 역할을 하는지 알아보니 굉장히 잘 짜여진 프로그램이라는 생각이 들었고 그래서 분석하는데 어렵지는 않았던 것 같다. 여러가지 장점들 때문에 큰 프로그램을 설계할 때는 특히 더 MVC 모델을 이용하여 객체 지향 설계를 해야 할 것 같다는 생각이 들었다. 또한, 사전에 각 클래스와 함수 역할을 명확히 하여 구현을 해야 쉽게 할 수 있을 것 같았다. 마지막으로, Enum 클래스는 익숙하지 않아 잘 사용하지 않았는데 이번 기회에 다시한번 공부하게 되어 많이 알게 되어 유익했다.

4. 소스코드

```
public class _DS01_Main_201702039_오명주 {  
    public static void main(String[] args) {  
        AppController appController = new AppController();  
        // AppController가 실질적인 main class  
        appController.run();  
    }  
}
```

[Main클래스]

```
public class AppView {  
    // 비공개 상수, 변수  
    private static Scanner scanner = new Scanner(System.in); // scanner import하여 입력받을  
  
    // 생성자 : 객체 생성할 일 없음  
    private AppView() {  
    }  
  
    // 입력 관련 함수  
    public static int inputOrder() {  
        int inputValue = scanner.nextInt(); // scanner를 이용하여 정수를 입력받는다  
        return inputValue; // 입력받은 정수를 return한다  
    }  
  
    // 출력 관련 함수  
    // 한줄을 출력하는 함수 (한줄이 띄워지지않는다)  
    public static void output(String message) {  
        System.out.print(message); // 입력받은 message를 출력한다  
    }  
  
    // 한줄을 출력하는 함수 (한줄이 띄워진다)  
    public static void outputLine(String message) {  
        System.out.println(message); // 입력받은 message를 출력한다  
    }  
  
    public static void outputTitleWithOrder(int order) {  
    }  
  
    // 마방진에 줄을 나타내는 함수  
    public static void outputRowNumber(int number) {  
        System.out.printf("[%3d] ", number); // number를 입력받아 주어진 형식으로 출력한다  
    }  
  
    // 마방진 수를 채우는 함수  
    public static void outputCellValue(int value) {  
        System.out.printf(" %3d ", value); // value를 입력받아 마방진 주어진 위치에 수를 출력한다  
    }  
}
```

[AppView 클래스]

```

public static final int MIN_ORDER = 3; // 마방진을 만들 수 있는 가장 작은 수
public static final int MAX_ORDER = 99; // 마방진을 만들 수 있는 가장 큰 수

// 비공개 변수들
private MagicSquare _magicSquare;

// 생성자
public AppController() {
    this._magicSquare = new MagicSquare(AppController.MAX_ORDER);
}

// 공개함수
public void run() {
    AppView.outputLine("<<< 마방진 풀이를 시작합니다 >>>");
    AppView.outputLine("");
    AppView.output("<? 마방진 자수를 입력하십시오(음수를 입력하면 종료합니다): >");

    int currentOrder = AppView.inputOrder(); // 메시지를 내보내고 자수를 입력받음
    OrderValidity currentValidity = OrderValidity.validityOf(currentOrder);
    while (currentValidity != OrderValidity.EndOfRun) { // 자수가 음수이면 프로그램 종료
        if (currentValidity == OrderValidity.Valid) { // 자수가 유효한지 검사
            AppView.outputTitleWithOrder(currentOrder);
            Board solvedBoard = this._magicSquare.solve(currentOrder);
            // _magicSquare 객체에 주어진 자수의 마방진을 풀도록 시킨다.
            // 결과로 마방진 판을 얻는다
            this.showBoard(solvedBoard); // 마방진을 화면에 보여준다
        } else {
            this.showOrderValidityErrorMessage(currentValidity);
        }
        AppView.outputLine("");
        AppView.output("<? 마방진 자수를 입력하십시오(음수를 입력하면 종료합니다): >");
        currentOrder = AppView.inputOrder(); // 다음 마방진을 위해 자수를 입력받음
        currentValidity = OrderValidity.validityOf(currentOrder);
    }
    AppView.outputLine("");
    AppView.outputLine("<<< 마방진 풀이를 종료합니다 >>>");
}

// 자수가 유효한지 알려주는 함수
private void showOrderValidityErrorMessage(OrderValidity orderValidity) {
    // enum에서 설정한 유효성을 바탕으로 오류검출
    switch (orderValidity) {
        case TooSmall:
            AppView.outputLine("[오류]자수가 너무 작습니다. " + AppController.MIN_ORDER + "보다 크거나 같아야 합니다.");
            break;
        case TooLarge:
            AppView.outputLine("[오류]자수가 너무 큼니다. " + AppController.MAX_ORDER + "보다 작거나 같아야 합니다.");
            break;
        case NotOddNumber:
            AppView.outputLine("[오류]자수가 짝수입니다. 홀수이어야 합니다.");
            break;
        default:
            break;
    }
}

// 마방진을 보여주는 함수
private void showBoard(Board board) {
    CellLocation currentLoc = new CellLocation();
    this.showTitleForColumnIndexes(board.order());
    for (int row = 0; row < board.order(); row++) {
        AppView.outputLineNumber(row); // AppView의 마방진 줄 출력함수를 이용하여 출력
        for (int col = 0; col < board.order(); col++) {
            currentLoc.setRow(row);
            currentLoc.setCol(col);
            AppView.outputCellValue(board.cellValue(currentLoc)); // AppView의 마방진 수 출력함수 이용하여 출력
        }
        AppView.outputLine("");
    }
}

// 마방진의 결과 열을 보여주는 함수 [ 0]과 같은.
private void showTitleForColumnIndexes(int order) {
    AppView.output(""); // 처음 공백
    for (int col = 0; col < order; col++) {
        AppView.output(String.format(" [%3d]", col));
    }
    AppView.outputLine("");
}
}

```

[AppController 클래스]

```

public class Board {

    // 상수
    private static int EMPTY_CELL = -1;

    // private instance variables
    private int _order;
    private int[][] _cells;

    // Getters / Setters
    public int order() { // 마방진 자수를 얻는다
        return this._order;
    }

    private void setOrder(int newOrder) { // 마방진 자수를 주어진 값으로 설정한다
        this._order = newOrder;
    }

    private int[][] cells() { // _cells 객체를 얻는다
        return this._cells;
    }

    private void setCells(int[][] newCells) { // _cells를 주어진 객체로 설정한다
        this._cells = newCells;
    }

    // 기본 생성자
    public Board(int givenOrder) {
        this.setOrder(givenOrder);
        this.setCells(new int[givenOrder][givenOrder]);
        for(int row = 0; row < givenOrder; row++) {
            for(int col = 0; col < givenOrder; col++) {
                this.setCellValue(row, col, Board.EMPTY_CELL);
            }
        }
    }

    // 비어 있으면 true, 아니면 false를 얻는다
    public boolean isEmpty() {
        return (this.cellValue(0, 0) == EMPTY_CELL);
    }

    public int cellValue(CellLocation location) {
        // 주어진 location의 cell값을 얻는다
        return this._cells()[location.row()][location.col()];
    }

    public void setCellValue(CellLocation location, int newCellValue) {
        // 주어진 location의 cell에 주어진 value를 넣는다
        this._cells()[location.row()][location.col()] = newCellValue;
    }

    private void setCellValue(int row, int col, int newCellValue) {
        // 이 method는 class 내부에서만 사용된다
        // 주어진 위치 (row, col)의 cell에 주어진 값 value를 넣는다
        this._cells()[row][col] = newCellValue;
    }
}

```

[Board 클래스]

```

public class CellLocation {

    // 상수
    private static final int UNDEFINED_INDEX = -1;

    // private instance variables
    private int _row;
    private int _col;

    // 기본 생성자 : Cell 좌표가 주어지지 않는다
    public CellLocation() {
        // Cell좌표가 주어지지 않으면 (-1, -1)로 설정하기로 한다
        this.setRow(UNDEFINED_INDEX);
        this.setCol(UNDEFINED_INDEX);
    }

    // Cell 좌표가 주어지는 생성자
    public CellLocation(int givenRow, int givenCol) {
        this.setRow(givenRow);
        this.setCol(givenCol);
    }

    // Getter / Setter
    // 위치의 row 좌표를 입력받아 설정한다
    public void setRow(int newRow) {
        this._row = newRow;
    }
    // 위치의 row 좌표를 반환한다
    public int row() {
        return this._row;
    }
    // 위치의 col 좌표를 입력받아 설정한다
    public void setCol(int newCol) {
        this._col = newCol;
    }
    // 위치의 col 좌표를 반환한다
    public int col() {
        return this._col;
    }
}

```

[CellLocation 클래스]

```

// 프로그램의 유효성을 판단하는 enum
public enum OrderValidity {
    EndOfRun, Valid, TooSmall, TooLarge, NotOddNumber;

    public static OrderValidity validityOf(int order) {
        // 만약 입력받은 자수가 음수라면 EndOfRun으로 설정
        if (order < 0) {
            return OrderValidity.EndOfRun;
        }
        // 만약 입력받은 자수가 설정한 최소값보다 작으면 TooSmall로 설정
        else if (order < ApplicationController.MIN_ORDER) {
            return OrderValidity.TooSmall;
        }
        // 만약 입력받은 자수가 설정한 최대값보다 크면 TooLarge로 설정
        else if (order > ApplicationController.MAX_ORDER) {
            return OrderValidity.TooLarge;
        }
        // 만약 입력받은 자수가 짝수라면 NotOddNumber로 설정
        else if ((order % 2) == 0) {
            return OrderValidity.NotOddNumber;
        }
        // 만약 유효하다면 Valid로 설정
        else {
            return OrderValidity.Valid;
        }
    }
}

```

[OrderValidity 클래스]

```

public class MagicSquare {
    private static final int DEFAULT_MAX_ORDER = 99;
    private int _maxOrder;

    // Getters/Setters
    public int maxOrder() {
        return this._maxOrder;
    }

    private void setMaxOrder(int newMaxOrder) {
        this._maxOrder = newMaxOrder;
    }

    // 기본 생성자
    public MagicSquare() {
        this.setMaxOrder(MagicSquare.DEFAULT_MAX_ORDER);
    }

    // 최대 자수를 사용자가 지정하는 생성자
    public MagicSquare(int givenMaxOrder) {
        this.setMaxOrder(givenMaxOrder);
    }

    public Board solve(int anOrder) {
        if (OrderValidity.validityOf(anOrder) != OrderValidity.Valid) {
            return null;
        } else {
            Board board = new Board(anOrder);
            // 자수와 함께 Board 객체 생성자를 call 하여, Board 객체를 생성한다
            CellLocation currentLoc = new CellLocation(0, anOrder / 2);
            // 출발 위치 보드의 맨 왼쪽 맨 가운데를 현재의 위치로 설정
            CellLocation nextLoc = new CellLocation();
            board.setCellValue(currentLoc, 1);
            // 보드의 출발 위치 예 1 을 채운다
            int lastValue = anOrder * anOrder;
            for (int cellValue = 2; cellValue <= lastValue; cellValue++) {
                // 단계 1: <현재 위치>로부터 <다음 위치>인 "오른쪽 위" 위치를 계산한다
                // 만약 "오른쪽 위" 위치가 주어진 행과 열 을 벗어나면 다시 조정한다
                if (currentLoc.row() - 1 < 0 && currentLoc.col() + 1 >= anOrder) {
                    nextLoc.setRow(currentLoc.row() - 1 + anOrder);
                    nextLoc.setCol(currentLoc.col() + 1 - anOrder);
                }
                // 만약 "오른쪽 위" 위치가 주어진 행을 벗어나면 가장 아래쪽으로 위치를 다시 조정한다
                else if (currentLoc.row() - 1 < 0) {
                    nextLoc.setRow(currentLoc.row() - 1 + anOrder);
                    nextLoc.setCol(currentLoc.col() + 1);
                }
                // 만약 "오른쪽 위" 위치가 주어진 열을 벗어나면 가장 좌측으로 위치를 다시 조정한다.
                else if (currentLoc.col() + 1 >= anOrder) {
                    nextLoc.setRow(currentLoc.row() - 1);
                    nextLoc.setCol(currentLoc.col() + 1 - anOrder);
                }
                // 만약 "오른쪽 위" 위치가 주어진 행과 열을 벗어나지 않는다면 좌표를 설정한다
                else {
                    nextLoc.setRow(currentLoc.row() - 1);
                    nextLoc.setCol(currentLoc.col() + 1);
                }
                // 단계 2: <다음 위치>가 채워져 있으면
                // <다음 위치>를 <현재 위치>의 바로 한 줄 아래 칸 위치로 수정한다
                if (!board.cellIsEmpty(nextLoc)) {
                    nextLoc.setRow(currentLoc.row() + 1);
                    nextLoc.setCol(currentLoc.col());
                }
                // 단계 3: <다음 위치>를 새로운 현재 위치 로 한다
                currentLoc.setRow(nextLoc.row());
                currentLoc.setCol(nextLoc.col());
                // 단계 4: 새로운 현재 위치 에 number 값을 넣는다
                board.setCellValue(currentLoc, cellValue);
            }
            return board;
        }
    }
}

```

[MagicSquare 클래스]