자료구조: 2022년 1학기 [실습]

제 1, 2 주:

마방진

강 지 훈
jhkang@cnu.ac.kr
충남대학교 컴퓨터융합학부

실습 목표

실습 목표

- Model-View-Controller 설계 방식을 이해하고 적용
- 주어진 문제를 이해하여 풀고, Java 언어로 구현하는 방법
- 과제에 필요한 Java 의 표현을 이해하고 적용
  - 2 차원 배열
  - Enum Class

"마방진 이란?"

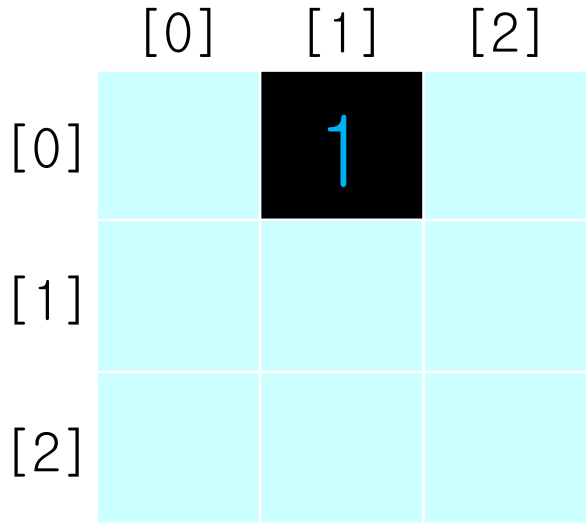
마방진 (Magic Square)

- 주어진 조건
  - 가로 세로의 크기가 동일한 정사각형 판이 주어져 있다.
  - 한 변의 크기를 마방진 문제의 차수(order) 라고 한다.
  - 차수는 3 보다 크거나 같은 홀수이다.
- 문제
  - 차수가 N 일 때, 1부터 N*N 까지의 정수를 한번씩 모두 사용하여 판을 채우되 가로, 세로, 대각선의 합이 모두 동일하게 채워진 판을 찾는다.
  - 예: 차수(order)가 3 인 마방진
    - 1부터 9 (=3x3) 까지의 정수를 한번씩 사용하여, 판을 채운다.
    - 가로, 세로, 대각선의 합이 모두 15 이다.

	[0]	[1]	[2]
[0]	8	1	6
[1]	3	5	7
[2]	4	9	2

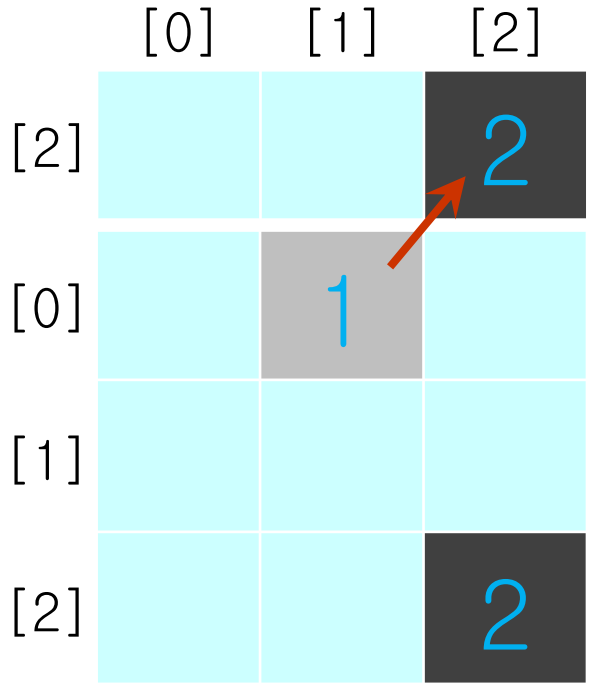
출발점

- 출발점은 맨 윗 줄의 한 가운데:
  - 차수가 3 인 경우, [0][1] 의 위치



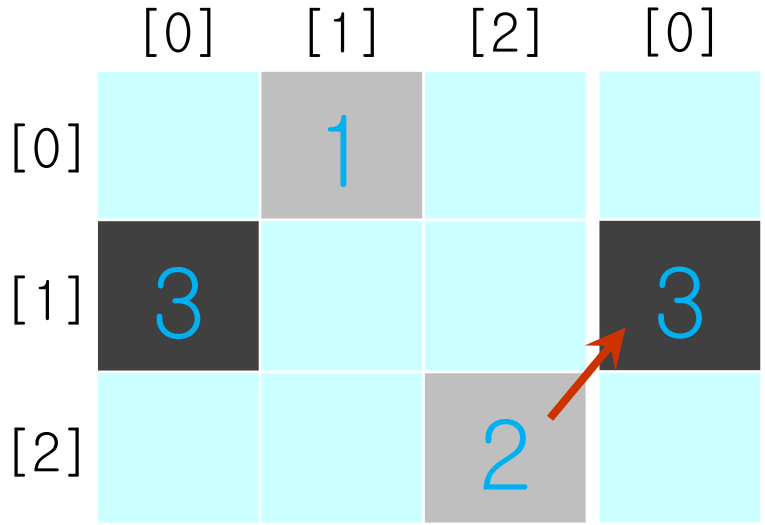
다음 채울 곳

- 항상 오른쪽 위로 올라간다.
- 판을 벗어나게 되면?
  - '1' 을 넣은 위치에서 오른쪽 위로 올라가면 판의 위쪽으로 벗어난다.
  - 판을 위아래가 붙은 마치 원통형인 것처럼 생각하면, 그 다음 '2' 의 위치는 [2][2] 가 된다.



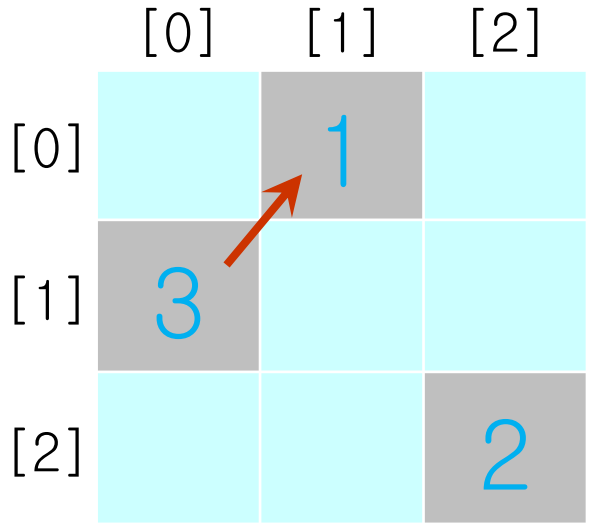
다음 채울 곳

- 항상 오른쪽 위로 올라간다.
- [2][2] 위치에서 오른쪽 위로 올라가면 판의 오른쪽으로 벗어난다.
  - 판을 좌우로 붙은 원통형인 것처럼 생각하면, 그 다음 '3' 의 위치는 [1][0]가 된다.



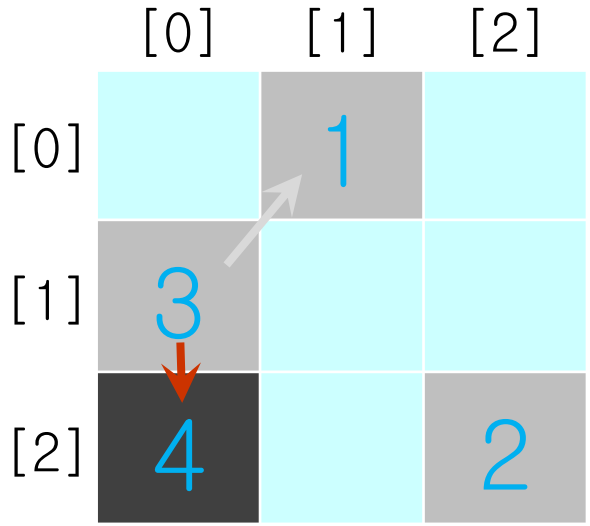
다음 채울 곳이 이미 채워져 있으면?

- 다음 위치는?
  - '3' 을 채운 [1][0] 의 오른쪽 위 위치인 [0][1] 에는 이미 '1' 이 채워져 있다.



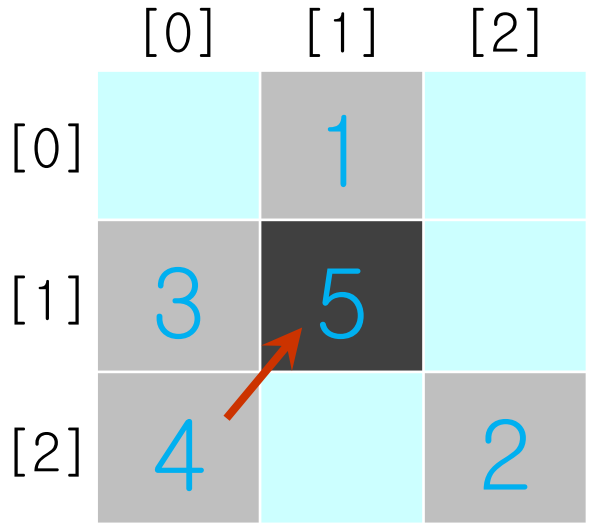
다음 채울 곳이 이미 채워져 있으면?

- 바로 한 칸 아래가 다음 위치이다.
  - '3' 을 채운 [1][0]의 오른쪽 위 위치인 [0][1] 에는 이미 '1'이 채워져 있다.
  - 그러므로 '4' 를 채울 다음 위치는 [2][0] 이다.



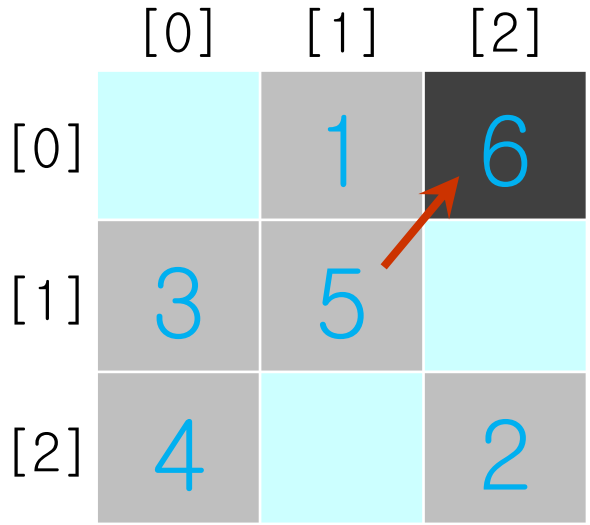
다음 채울 곳

- 오른쪽 위로 가자!
  - 다음 위치 [1][1] 은 비어있다.
  - 이곳에 '5' 를 채운다.



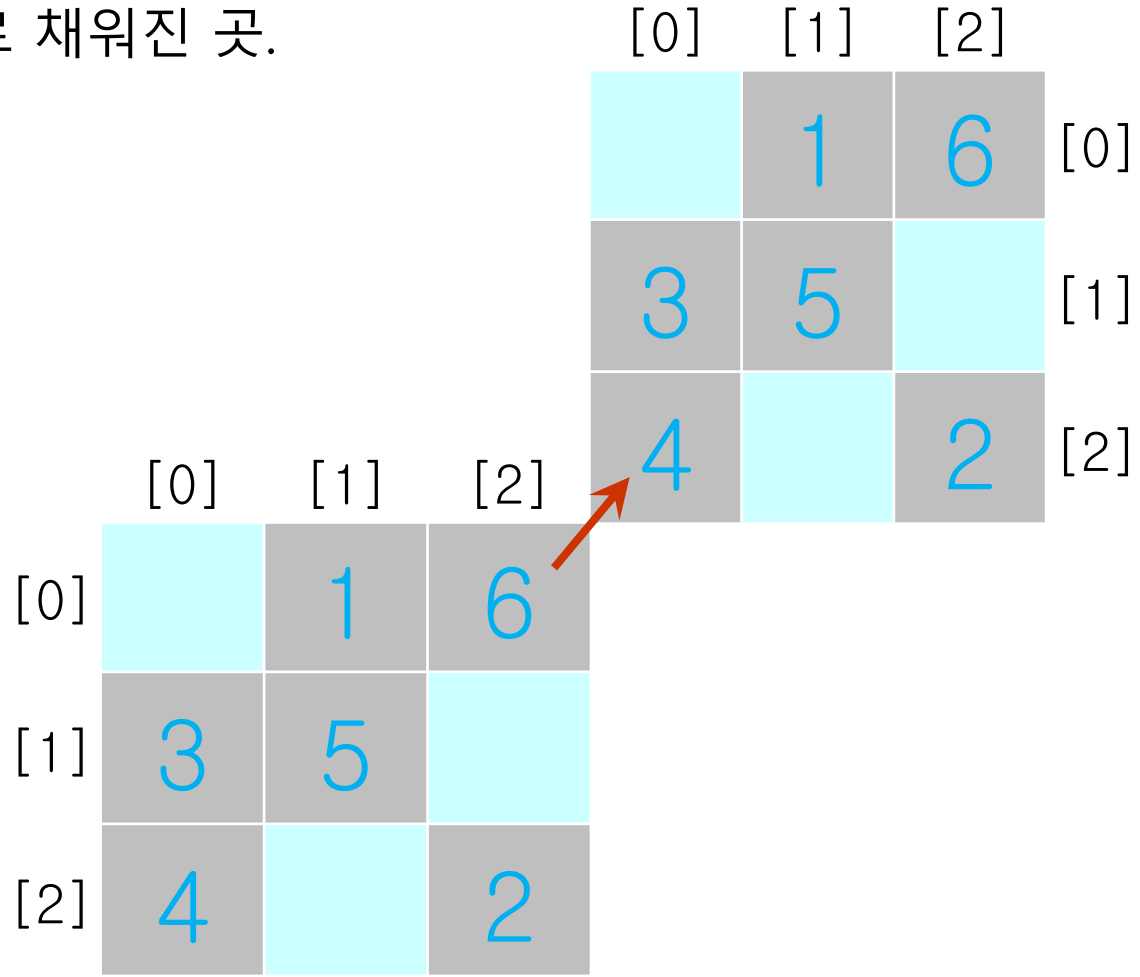
다음 채울 곳

- 오른쪽 위로 가자!
  - 다음 위치 [0][2] 는 비어있다.
  - 이곳에 '6' 을 채운다.



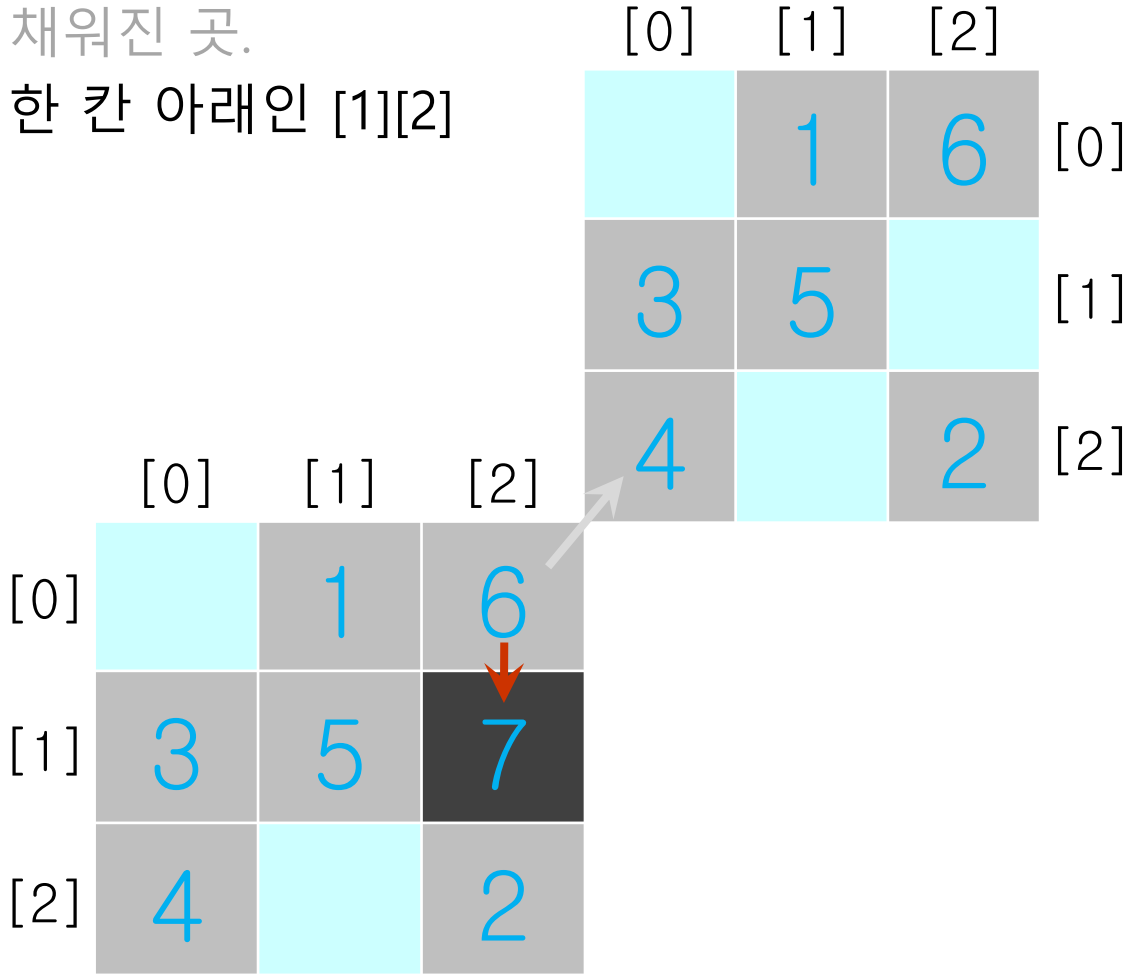
다음 채울 곳

- 오른쪽 위로 가자!
  - 상하좌우가 붙어있다고 생각하면, 다음 위치는 [2][0].
  - 이곳은 이미 '4' 로 채워진 곳.



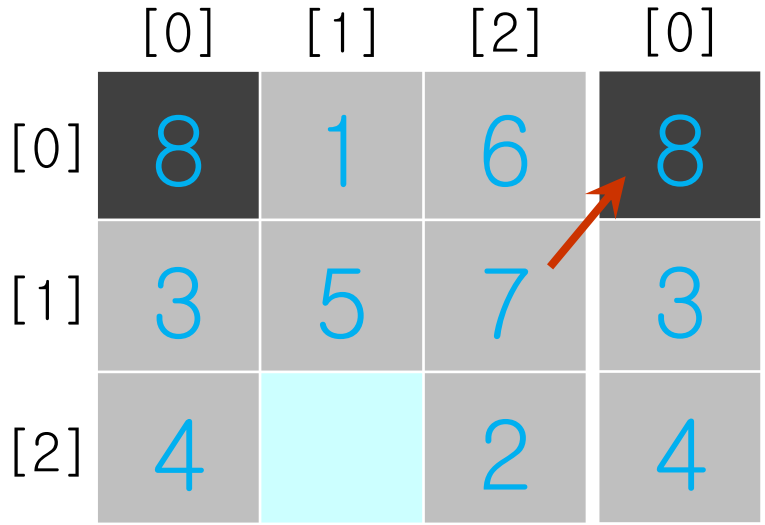
다음 채울 곳

- 채워져 있으므로 바로 아래로!
  - 상하좌우가 붙어있다고 생각하면, 다음 위치는 [2][0].
  - 이곳은 이미 '4'로 채워진 곳.
  - 다음 위치는 바로 한 칸 아래인 [1][2]



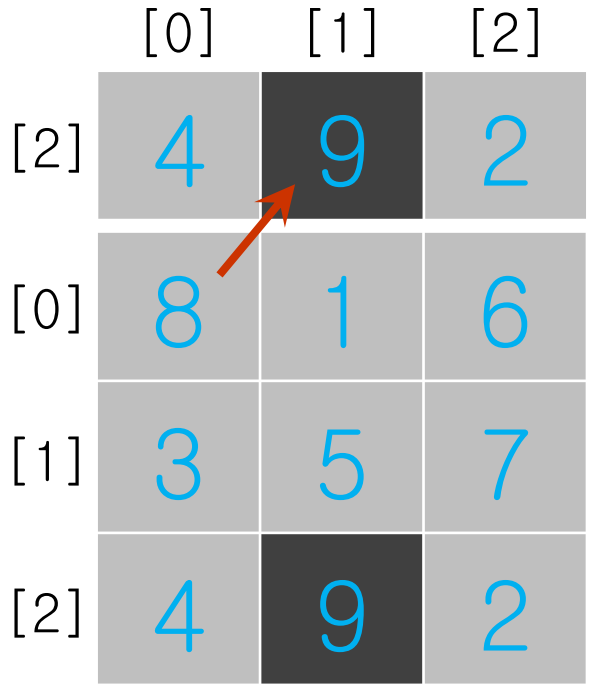
다음 채울 곳

- 오른쪽 위로 가자!
  - 다음 위치는 [0][0].
  - 이곳에 '8' 을 채운다.



다음 채울 곳

- 오른쪽 위로 가자!
  - 다음 위치는 [2][1].
  - 이곳에 '9' 를 마지막으로 채운다.



채우기 종료!

- '1' 부터 '9' 까지의 정수를 모두 채웠다.
  - 가로, 세로, 대각선의 합은 모두 동일하게 15 이다.

	[0]	[1]	[2]
[0]	8	1	6
[1]	3	5	7
[2]	4	9	2

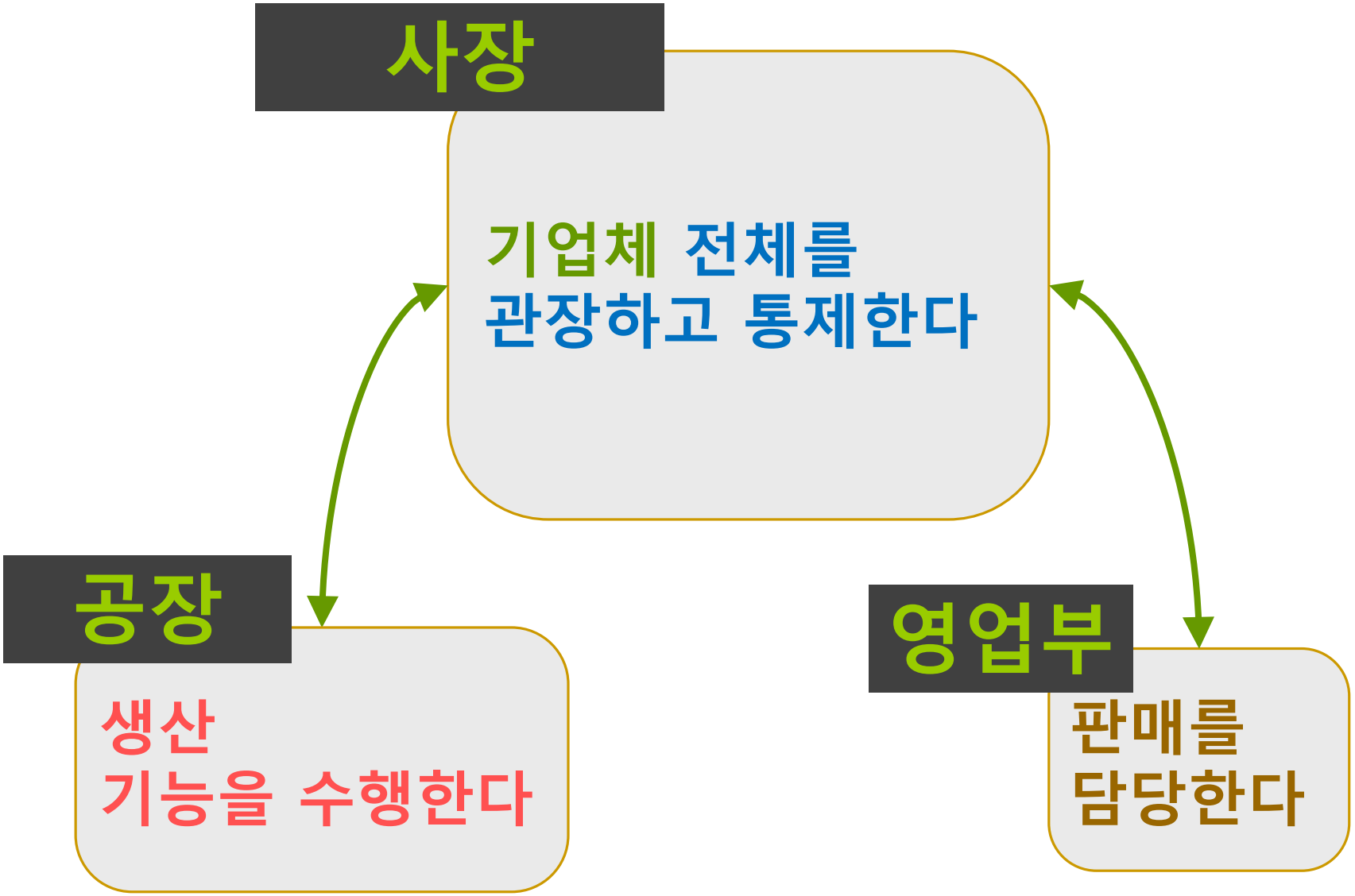
요약

- 차수가 홀수이면 항상 해결 가능
- 차수가 짝수이면?

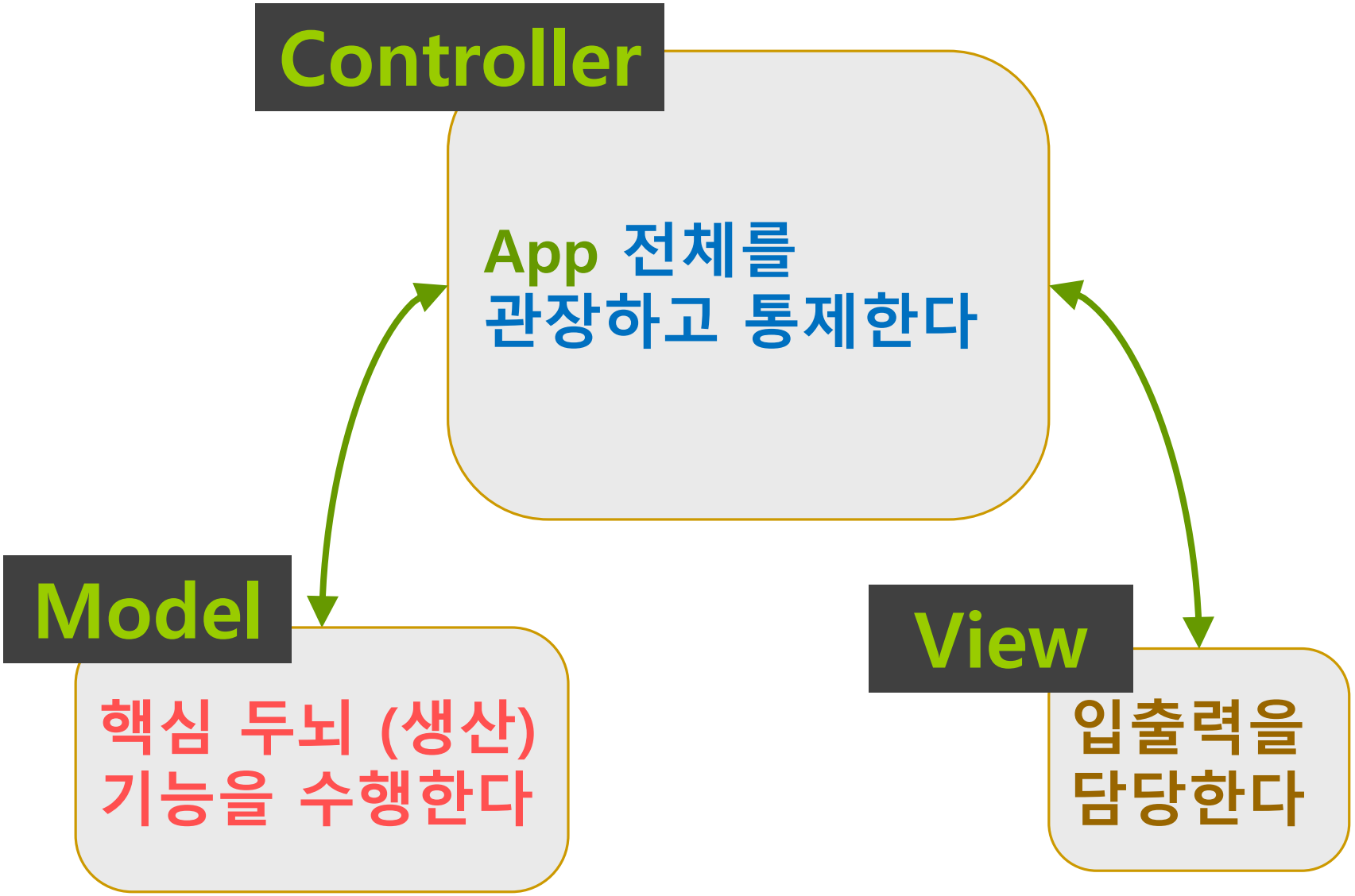
- 문제의 인식, 그리고 알고리즘으로, 프로그램으로!
  - 문제의 인식
    - 문제를 분석하고 이해
  - 알고리즘으로!
    - 규칙의 발견, 그리고 정확히 표현
  - 프로그램으로!
    - 주어진 언어 (우리는 Java 언어)로 적절하게 표현
    - 객체의 인식

"Model-View-Controller"

App 에서의 역할 구분

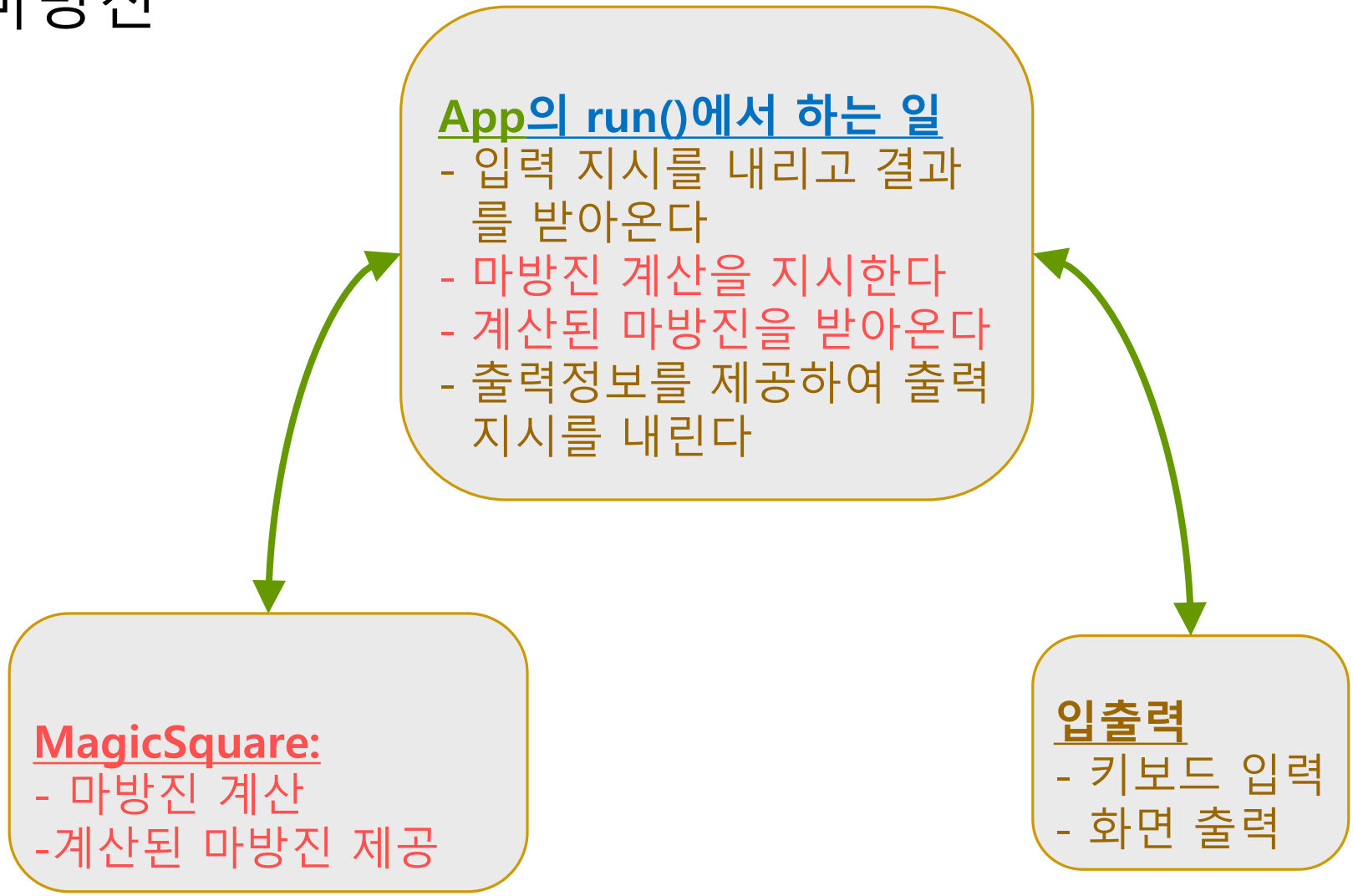


App 에서의 역할 구분

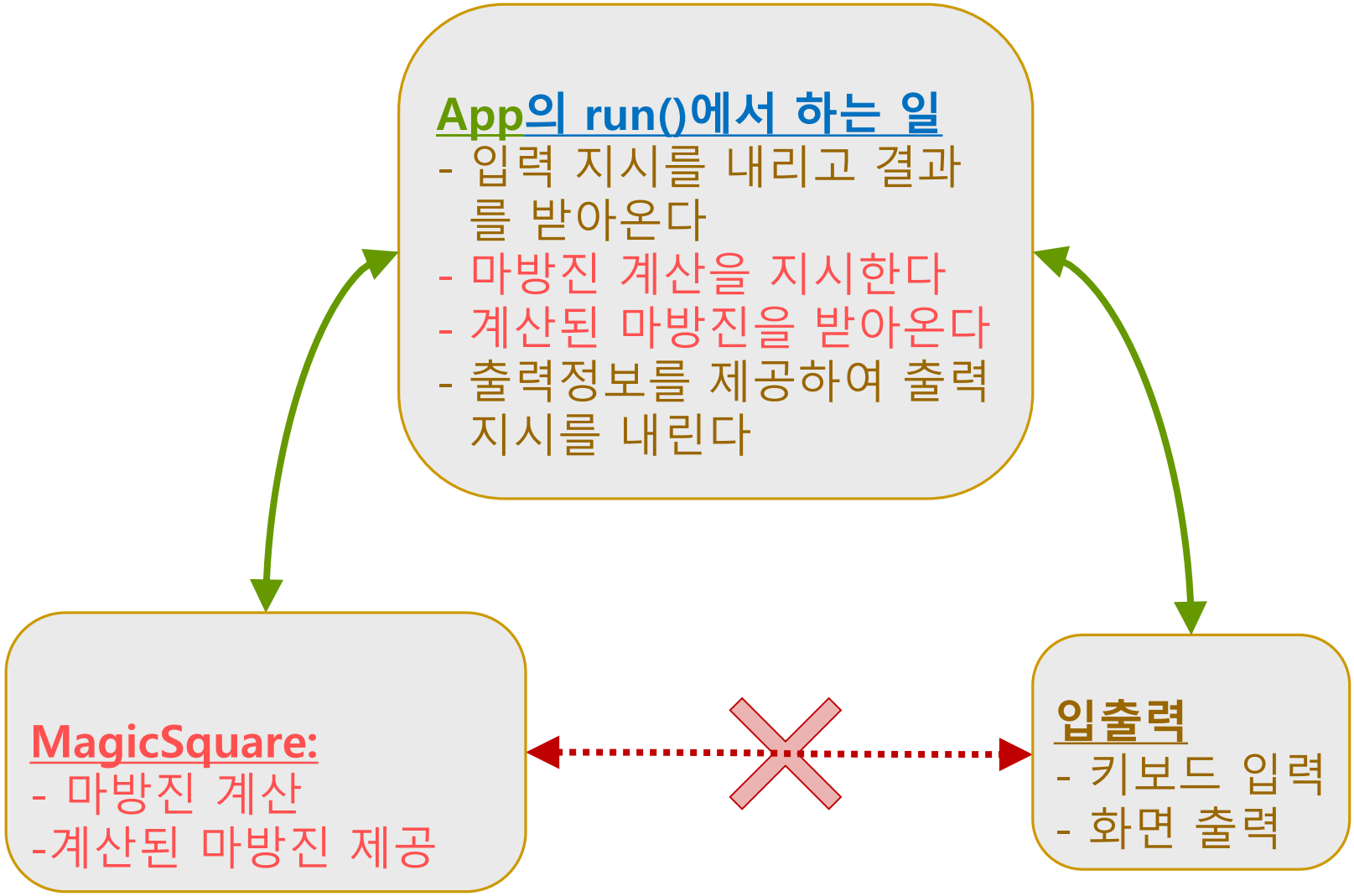

Controller
App 전체를
관장하고 통제한다
Model
핵심 두뇌 (생산)
기능을 수행한다
View
입출력을
담당한다

모든 프로그램은 MVC 형태로

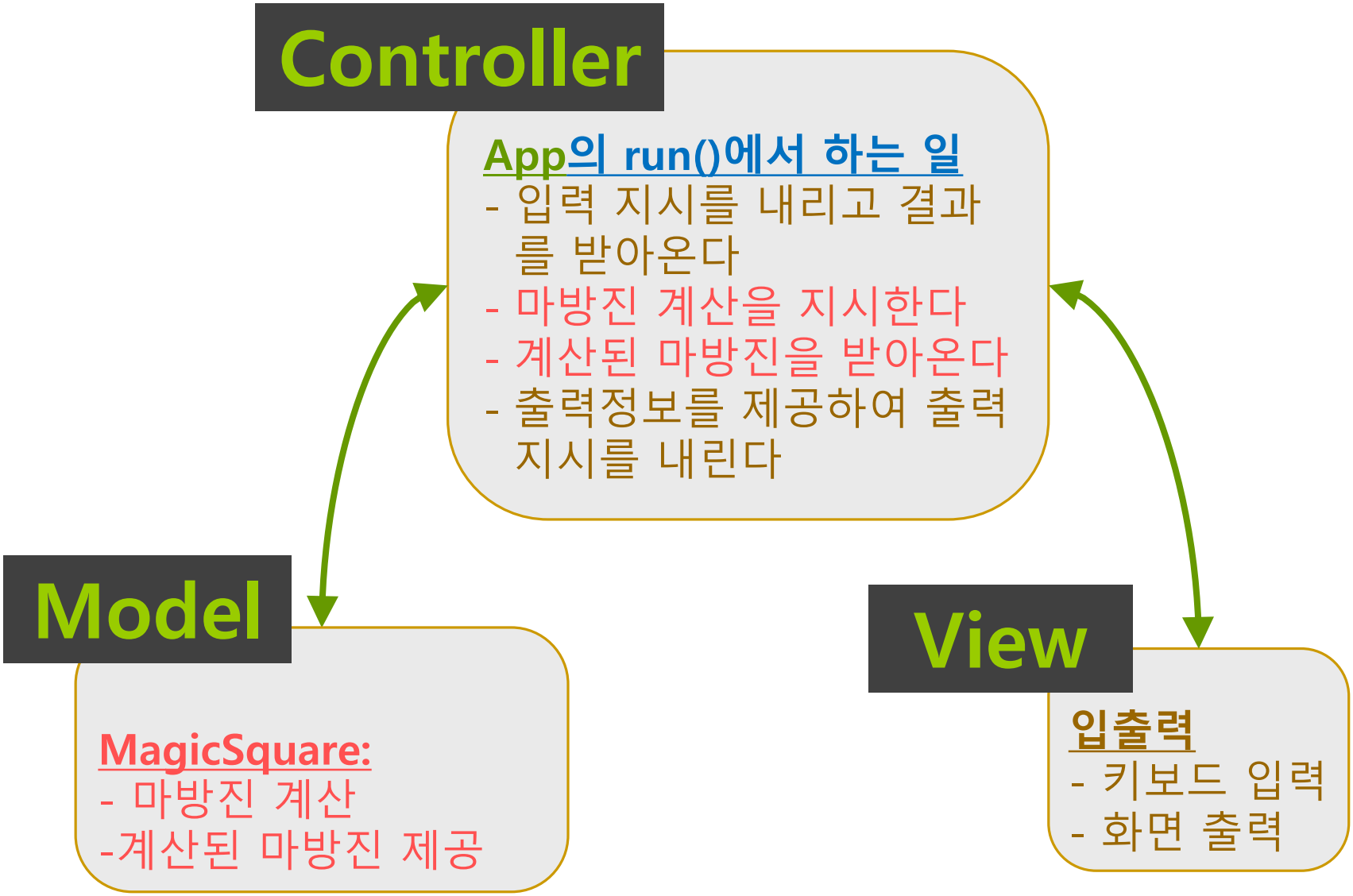
- 마방진



부하 직원이 상사 몰래?


App의 run()에서 하는 일
- 입력 지시를 내리고 결과를 받아온다
- 마방진 계산을 지시한다
- 계산된 마방진을 받아온다
- 출력정보를 제공하여 출력 지시를 내린다
MagicSquare:
- 마방진 계산
-계산된 마방진 제공
입출력
- 키보드 입력
- 화면 출력

MVC


Controller
App의 run()에서 하는 일
- 입력 지시를 내리고 결과를 받아온다
- 마방진 계산을 지시한다
- 계산된 마방진을 받아온다
- 출력정보를 제공하여 출력 지시를 내린다
Model
MagicSquare:
- 마방진 계산
-계산된 마방진 제공
View
입출력
- 키보드 입력
- 화면 출력

각각을 class 로!

- 마방진 프로그램
  - Controller: Class “AppController”
  - Model: Class “MagicSquare”
  - View: Class “AppView”

"for 문"

"for" 문

- while 문과 동등한 표현: 다음의 두 문장은 정확히 동등

```
int  sum = 0 ;
{
    int  count = 0 ;
    while ( count < size) {
        sum = sum + count ;
        count++
    }
}
```

⟺

```
int  sum = 0 ;

for ( int  count = 0 ; count < size ; count++) {
    sum = sum + count ;

}
```

  - 두 변수 "**sum**" 과 "**count**" 각각의 scope 는?

- 변수의 **scope (유효 영역)**
  - 프로그램 코드에서 변수가 선언에 의해 존재하는 위치부터, 선언을 포함하는 코드를 닫아주는 괄호 "}" 까지의 코드 영역을, 그 변수의 scope 라고 한다.
  - 모든 변수는 자신의 유효 영역 안에서만 유효하고 또한 존재함
  - (예) 함수의 매개변수의 scope 는 함수 내부:
    - 함수의 시작과 함께 존재하고, 함수를 종료함과 동시에 유효하지 않으며, 또한 더 이상 존재하지 않음.

과제에서 해결할 문제

문제

- 주어진 차수에 합당한 마방진 문제를 풀어, 화면에 보여준다.

입출력

- 입력
  - 매번 마방진 차수를 입력 받는다:
  - 음수이면 "<<<마방진 풀이를 종료합니다>>>" 라는 메시지를 내보내고 프로그램을 종료한다
  - 음수가 아니면 차수의 오류 검사를 다음과 같이 한다:
    - 차수가 3 보다 작으면 "[오류] 차수가 너무 작습니다. 3 보다 크거나 같아야 합니다." 라는 메시지를 내보낸다.
    - 차수가 99 보다 크면 "[오류] 차수가 너무 큽니다. 99 보다 작거나 같아야 합니다." 라는 메시지를 내보낸다.
    - 차수가 짝수 이면 "[오류]: 차수가 짝수입니다. 홀수이어야 합니다." 라는 메시지를 내보낸다.

- 출력
  - 매번 입력된 차수에 대한 마방진 풀이 결과를 출력한다.

출력의 예

```
<<< 마방진 풀이를 시작합니다 >>>

? 마방진 차수를 입력하시오 (음수를 입력하면 종료합니다): 2
[오류] 차수가 너무 작습니다. 3 보다 크거나 같아야 합니다.

? 마방진 차수를 입력하시오 (음수를 입력하면 종료합니다): 5
! Magic Square Board: Order 5
       [ 0 ]  [ 1 ]  [ 2 ]  [ 3 ]  [ 4 ]
[ 0 ]     17     24      1      8     15
[ 1 ]     23      5      7     14     16
[ 2 ]      4      6     13     20     22
[ 3 ]     10     12     19     21      3
[ 4 ]     11     18     25      2      9

? 마방진 차수를 입력하시오. (음수를 입력하면 종료합니다): 6
[오류] 차수가 짝수입니다. 홀수이어야 합니다.

? 마방진 차수를 입력하시오(음수를 입력하면 종료합니다): 100
[오류] 차수가 너무 큽니다. 99 보다 작거나 같아야 합니다.

? 마방진 차수를 입력하시오. (음수를 입력하면 종료합니다): 3
! Magic Square Board: Order 3
       [ 0 ]  [ 1 ]  [ 2 ]
[ 0 ]      8      1      6
[ 1 ]      3      5      7
[ 2 ]      4      9      2
```

```
? 마방진 차수를 입력하시오. (음수를 입력하면 종료합니다): 7
! Magic Square Board: Order 7
       [ 0 ]  [ 1 ]  [ 2 ]  [ 3 ]  [ 4 ]  [ 5 ]  [ 6 ]
[ 0 ]     30     39     48      1     10     19     28
[ 1 ]     38     47      7      9     18     27     29
[ 2 ]     46      6      8     17     26     35     37
[ 3 ]      5     14     16     25     34     36     45
[ 4 ]     13     15     24     33     42     44      4
[ 5 ]     21     23     32     41     43      3     12
[ 6 ]     22     31     40     49      2     11     20

? 마방진 차수를 입력하시오. (음수를 입력하면 종료합니다): -1

<<< 마방진 풀이를 종료합니다 >>>
```

이 과제에서 필요한 class 는?

- class AppController
- class AppView
- Model
  - class MagicSquare
  - class Board
  - class CellLocation
  - enum OrderValidity

프로그램이 시작하는 곳을 포함하는 특별한 class

"main()" 을 포함하는 특별한 class

```
public class _DS01_Main_학번_이름 {
    public static void main (String[] args)
    {
        AppController  appController = new AppController() ;
            // AppController 가 실질적인 main class 이다.
        appController.run() ;
            // 여기 main() 에서는 앱 실행이 시작되도록 해주는 일이 전부이다.
    }
}
```

main() 은 어떻게?

- main() 을 위한 class 에서는 App에서 필요로 하는 어떠한 변수도, 상수도, 함수도, 선언하지 않는다.
- 이제 실제 main 의 역할로서 필요하다고 판단되는 모든 것은 class "AppController" 안에 선언한다.
- main() 함수는 단지 아래의 코드만 필요하다.

```
public static void main (String[] args)
{
    AppController  appController = new AppController() ;
    appController.run() ; // 여기에서는 앱이 실행되도록 해주는 일이 전부이다.
}
```

- 앞으로 모든 Java 프로그램은 이런 형태로 작성한다.
  - Java 프로그램 뿐만 아니라 어떤 언어로 된 프로그램도 이와 같이 작성한다.
  - MVC 모델도 철저히 지킨다.

Class "AppController"

AppController: 생성자, 공개함수

- 생성자
  - public AppController() ;
- 공개함수
  - public void run() ; // 유일한 공개함수

Class "AppController"의 구현

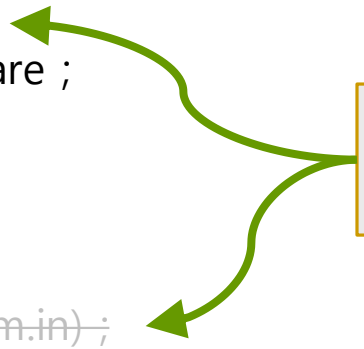
AppController: 상수, 인스턴스 변수, 생성자

```
public class AppController {
  // 공개 상수
  public static final int MIN_ORDER = 3 ;
  public static final int MAX_ORDER = 99 ;

  // 비공개 변수들
  private Scanner          _scanner ;
  private MagicSquare      _magicSquare ;

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

  // 비공개함수의 구현
  ......

  // 공개함수의 구현
  ......
} // End of class "AppController"
```

다른 Class 에서도 사용하므로, public 으로 선언한다.

AppController: scanner 는 어디에?

```
public class AppController {
   // 공개 상수
   public static final int MIN_ORDER = 3 ;
   public static final int MAX_ORDER = 99 ;

   // 비공개 변수들
   private Scanner          _scanner ;
   private MagicSquare      _magicSquare ;

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

   // 비공개함수의 구현
   ......

   // 공개함수의 구현
   ......
} // End of class "AppController"
```



MVC 설계 원칙을 지키자:
scanner는 AppView 속으로!

AppController: run()

```
public class AppController {
    ......

    // 공개함수의 구현
    public void  run()  {
        AppView.outputLine ("<<< 마방진 풀이를 시작합니다 >>>") ;
        AppView.outputLine ("") ;

        int  currentOrder = AppView.inputOrder() ; // 메시지를 내보내고 차수를 입력 받음
        OrderValidity currentValidity = OrderValidity.validityOf(currentOrder);
        while (currentValidity != OrderValidity.EndOfRun) { // 차수가 음수이면 프로그램 종료
            if (currentValidity == OrderValidity.Valid ) { // 차수가 유효한지 검사
                AppView.outputTitleWithOrder (currentOrder) ;
                Board solvedBoard= this._magicSquare.solve (currentOrder) ;
                    // _magicSquare  객체에게 주어진 차수의 마방진을 풀도록 시킨다.
                    // 결과로 마방진 판을 얻는다
                this.showBoard (solvedBoard) ; // 마방진을 화면에 보여준다
            }
            else {
                this.showOrderValidityErrorMessage (currentValidity) ;
            }
            currentOrder = AppView.inputOrder() ; // 다음 마방진을 위해 차수를 입력 받음
            currentValidity = OrderValidity.validityOf(currentOrder);
        } // end while
        AppView.outputLine ("");
        AppView.outputLine ("<<< 마방진 풀이를 종료합니다 >>>") ;
    }
}
```

AppController: run() – while loop 의 구조

```
public class AppController {
    ......

    // 공개함수의 구현
    public void  run()  {
        AppView.outputLine ("<<< 마방진 풀이를 시작합니다 >>>") ;
        AppView.outputLine ("") ;

        int  currentOrder = AppView.inputOrder() ; // 메시지를 내보내고 차수를 입력 받음
        OrderValidity currentValidity = OrderValidity.validityOf(currentOrder);
        while (currentValidity != OrderValidity.EndOfRun) { // 차수가 음수이면 프로그램 종료
            if (currentValidity == OrderValidity.Valid ) { // 차수가 유효한지 검사
                AppView.outputTitleWithOrder (currentOrder) ;
                Board solvedBoard= this._magicSquare.solve (currentOrder) ;
                    // _magicSquare  객체에게 주어진 차수의 마방진을 풀도록 시킨다.
                    // 결과로 마방진 판을 얻는다
                this.showBoard (solvedBoard) ; // 마방진을 화면에 보여준다
            }
            else {
                this.showOrderValidityErrorMessage (currentValidity) ;
            }
            currentOrder = AppView.inputOrder() ; // 다음 마방진을 위해 차수를 입력 받음
            currentValidity = OrderValidity.validityOf(currentOrder);
        } // end while
        AppView.outputLine ("");
        AppView.outputLine ("<<< 마방진 풀이를 종료합니다 >>>") ;
    }
}
```

AppController: showOrderValidityErrorMessages()

```
private  void showOrderValidityErrorMessage (OrderValidity orderValidity)
{
      switch ( orderValidity ) {
            case TooSmall:
                  AppView.outputLine (
                        "[오류] 차수가 너무 작습니다. " + AppController.MIN_ORDER +
                        " 보다 크거나 같아야 합니다.") ;
                  break ;
            case TooLarge:
                  AppView.outputLine (
                        "[오류] 차수가 너무 큽니다. " + AppController.MAX_ORDER +
                        " 보다 작거나 같아야 합니다. ") ;
                  break ;
            case NotOddNumber:
                   AppView.outputLine ("[오류] 차수가 짝수입니다. 홀수이어야 합니다.") ;
                  break ;
            default:
                  break ;
      }
}
```

AppController: showBoard()

```
private  void showBoard (Board board)
{
    CellLocation  currentLoc = new CellLocation() ;
    this.showTitleForColumnIndexes (board.order()) ;
    for ( int row = 0 ; row < board.order() ; row++ ) {
        AppView.outputRowNumber (row) ;
        for ( int col = 0 ; col < board.order() ; col++ ) {
            currentLoc.setRow (row) ;
            currentLoc.setCol (col) ;
            AppView.outputCellValue (board.cellValue(currentLoc)) ;
        }
        AppView.outputLine("") ;
    }
}
```

AppController: showTitleForColumnIndexes()

```
private void showTitleForColumnIndexes (int order) {
    AppView.output("      ") ; // 빈칸 6 개
    for (int col = 0 ; col < order ; col++ ) {
        AppView.output( String.format(" [%3d]", col) ) ;
    }
    AppView.outputLine("") ;
}
```

양식 " [%3d]" 가 의미하는 문자 배치 형태:
모두 6 칸의 문자열이 만들어진다.
(맨 앞 문자가 빈칸 임에 유의할 것)

이곳 3 칸에 정수 값이 들어간다

String.format (format, variable_list):
주어진 변수들 (variable_list)을 주어진 양식 (format)에 맞게 문자열 객체를 생성한다.
양식 "%3d" 는, 정수 값 하나를 3 칸 안에 배치함을 의미한다. 즉 "%"는 이에 상응하는 변수가 반드시 variable_list 에 주어져야 하며, "3" 은 어떤 변수의 값이 문자열 안에서 차지하는 공간을 지정하는 것이며, "d"는 상응하는 변수의 값이 정수형으로 문자열 안에 배치됨을 의미한다.

Class "AppView"

공개 함수

```
public AppView() {...} // 생성자

// 입력 관련 함수들
public static int    inputOrder() {...} // 마방진 차수를 입력 받아 얻는다

// 출력 관련 함수들
public static void  output (String message) {...}
public static void  outputLine (String message) {...}
public static void  outputTitleWithOrder (int order) {...}
public static void  outputRowNumber (int number) {...}
public static void  outputCellValue (int value) {...}
```

Class "AppView"의 구현

AppView: Scanner 선언, 생성자

```
public class AppView {
  // 비공개 상수/변수들
  private static Scanner     scanner = new Scanner(System.in) ;

  // 생성자: 객체를 생성할 일이 없음. 따라서 “private”.
  private  AppView ()
  {
  }

  // 공개함수의 구현
  ……
}
```

Class "AppView" 의 구현

```
public class AppView {
  // 비공개 상수/변수들
  .....

  // 생성자
  ......

  // 공개함수의 구현
  public static int  inputOrder () {
       ..... // 여기를 채우시오
  }
```

Class "AppView" 의 구현

- 출력 관련 함수들

```
public static void output (String message) {
      System.out.print (message) ;
}

public static void outputLine (String message) {
      System.out.println (message) ;
}

public static void  outputTitleWithOrder (int order) {
     ..... // 여기를 채우시오
}

public static void  outputRowNumber (int number) {
     System.out.printf ("[%3d] ", number) ;
}

public static void  outputCellValue (int value) {
     System.out.printf ("  %3d ", value) ;
}
```

Enum “OrderValidity”

Enum class

- Data type 은 값들의 집합이다.

```
OrderValidity = {
        EndOfRun,
        Valid,
        TooSmall,
        TooLarge,
        NotOddNumber
}
```

- 바로 이와 같은 자료형을 선언하게 해주는 것이 Java 에서는 enum class 이다.
- Enum data type 은 값들의 순서 집합이다.
  - 집합이면서, 또한 값들의 순서에 의미가 있다.
- Class 이므로:
  - 집합의 값들은 하나 하나가 객체이다.
    - (예) OrderValidity 의 EndOfRun 은, enum 값이면서 또한 객체이다.
  - Method (함수) 를 가질 수 있다.
    - Class Method / Instance Method 어느 것이나 가질 수 있다.

Enum "OrderValidity"

```
public enum OrderValidity {
        EndOfRun,
        Valid,
        TooSmall,
        TooLarge,
        NotOddNumber ;

        public static OrderValidity validityOf (int order) { // Class Method
            if (order < 0) {
                return OrderValidity.EndOfRun ;
            }
            else if (order < AppController.MIN_ORDER) {
                return OrderValidity.TooSmall ;
            }
            else if (order > AppController.MAX_ORDER) {
                return OrderValidity.TooLarge ;
            }
            else if ( (order % 2) == 0 ) {
                return OrderValidity.NotOddNumber ;
            }
            else {
                return OrderValidity.Valid ;
            }
        }

} // End of enum "OrderValidity"
```

Class "MagicSquare"

MagicSquare: 공개함수

```
public class MagicSquare {
        ……
        // 기본 생성자
        ……
        // 최대 차수를 사용자가 지정하는 생성자
        ……

        // 공개함수
        public int  maxOrder() {……}
            // 마방진의 현 상태의 최대 차수를 얻는다
        public Board  solve (int anOrder) {……}
            // 주어진 차수의 마방진을 풀도록 시킨다
            // 차수는 반드시 다시 검사해야 한다.
            // 풀어서 생성된 마방진 판을 얻는다. 오류가 있으면 null을 얻는다.

} // End of class "MagicSquare"
```

MagicSquare: 생성자

```
public class MagicSquare {
    private static int DEFAULT_MAX_ORDER = 99 ;

    private int        _maxOrder ;

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

    // 최대 차수를 사용자가 지정하는 생성자
    public  MagicSquare (int givenMaxOrder)
    {
        this._maxOrder = givenMaxOrder ;
    }
```

여러 가지 생성자를 만들어 놓고, 상황에 따라 골라서 사용할 수도 있다!
모든 생성자의 이름은 동일.

Class “MagicSquare”의 구현

MagicSquare: 상수, 인스턴스 변수

```
public class MagicSquare {
    private static final int DEFAULT_MAX_ORDER = 9 ;

    private int              _maxOrder ;
```

MagicSquare: getter

```
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 ;
    }
```

MagicSquare: 공개함수의 구현

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

    private int              _maxOrder ;

    // Getters / Setters
    public  int    maxOrder() {...}
    private void setMaxOrder(int newMaxOrder) {...}

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

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

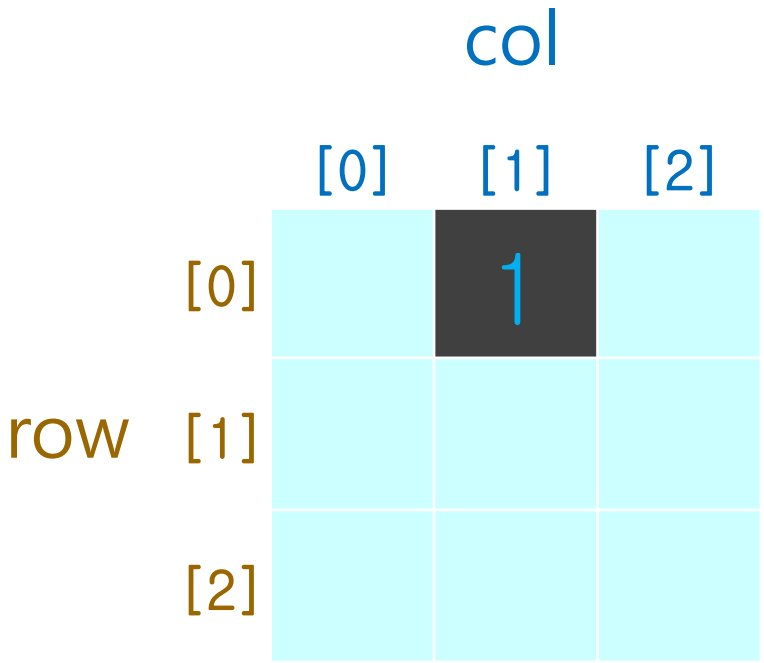
공개함수 "solve()"

```
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 가운데)를 현재의 위치로 설정한다.
            CellLocation  nextLoc = new CellLocation() ;

            board.setCellValue (currentLoc, 1) ;  // 보드의 <출발 위치>에 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
                  ..... // 여기를 채울 것
                  // 단계 2: <다음 위치> 가 채워져 있으면
                  // <다음 위치>를 <현재 위치>의 바로 한 줄 아래 칸 위치로 수정한다.
                  if ( ! board.cellIsEmpty (nextLoc) ) {
                        ..... // 여기를 채울 것
                  }
                  // 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 ;
      }
} // End of solve()
```

Class "CellLocation"

Class CellLocation

- Cell의 위치는 행(row) 과 열(col)로 표현할 수 있다.
- 이 두 값을 하나로 묶어서 객체로 만든다



CellLocation: 생성자, 공개함수

- 생성자:
  - public CellLocation() ;
    - 단지 객체만 생성한다
  - public CellLocation(int givenRow, int givenCol) ;
    - 주어진 좌표 (givenRow, givenCol) 을 갖는 객체를 생성한다

- 공개함수
  - public void setRow (int newRow) ; // setter for row
  - public int row () ; // getter for row
  - public void setCol (int newCol) ; // setter for col
  - public int col () ; // getter for col

Class “CellLocation”의 구현

CellLocation: 인스턴스 변수, 생성자

```
public class CellLocation {
        // Constant
        private static final int UndefinedIndex = -1;

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

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

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

        // 공개함수
        ……
```

CellLocation: 공개함수

```
public class CellLocation {

        ......

        // Getter / Setter
        public void setRow (int newRow) {
            this._row   = newRow ;
        }
        public int row ()  {
            return this._row ;
        }

        public void setCol (int newCol) {
            this._col   = newCol ;
        }
        public int col() {
            return this._col ;
        }
} End of class "CellLocation"
```

Class “Board”

Class "Board"

- 마방진 판을 위한 class
- 매번 풀기 전에 차수에 맞게 새롭게 객체를 생성하여 사용한다.
  - 생성 시에 차수가 주어진다.
- 빈칸과 채워진 칸을 구분한다.
  - 빈칸은 EMPTY_CELL (-1) 값으로 채운다.
- 생성자
  - 판을 생성시키고, 모든 칸을 빈칸으로 설정한다.

	[0]	[1]	[2]
[0]	-1	-1	-1
[1]	-1	-1	-1
[2]	-1	-1	-1

Class Board: 생성자, 공개함수

- 생성자
  - public Board (int givenOrder);
    - 주어진 차수의 마방진 판을 생성한다.
- 공개함수
  - public int order() ;
    - 마방진 판의 차수를 얻는다.
  - public int cellValue (CellLocation location) ;
    - 주어진 위치 aLocation의 cell 값을 얻는다.
  - public int cellValue (int row, int col) ;
    - 주어진 위치 (row, col) 의 cell 값을 얻는다.
  - public void setCellValue (CellLocation location, int cellValue) ;
    - 주어진 값 CellValue 를, 주어진 location 의 cell 에 넣는다.
  - public void setCellValue (int row, int col, int cellValue) ;
    - 주어진 값 cellValue 를, 주어진 위치 (row, col)의 cell 에 넣는다.
  - public boolean cellIsEmpty (CellLocation location) ;
    - 주어진 위치의 cell 이 비어 있는지 여부를 알려준다
    - 비어 있으면 true 를, 아니면 false 를 얻는다.

Class “Board”의 구현

Board: 상수, 인스턴스 변수

```
public class Board {

    // Constant
    private static int EMPTY_CELL = -1 ;

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

    // Getter/Setter
    public int  order() {  // 마방진 차수를 얻는다.
        return  this._order ;
    }
    private void setOrder (int newOrder) {  // 마방진 차수를 주어진 값으로 설정한다.
        this._order = newOrder ;
    }



    ......
```

Board: 생성자, 공개함수

```
public class Board {

    ……

    // 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) ;
            }
        }
    }

    // public methods
    public boolean cellIsEmpty (CellLocation location) {
        // 주어진 위치의 cell이 비어 있는지 여부를 알려준다
        // 비어 있으면 true, 아니면 false를 얻는다.
        return (this.cellValue(location) == EMPTY_CELL) ;
    ……
```

Board: 공개/비공개 함수

```
public class Board {
        ......

        private void  setCells (int[][] newCells) {
              this._cells = newCells ;
        }

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

        public void  setCellValue (CellLocation location, int value) {
              // 주어진 location 의 cell 에 주어진 value 를 넣는다.
              this._cells [location.row()][location.col()] = value ;
        }

        private void  setCellValue (int row, int col, int value) {
              // 이 method 는 class 내부에서만 사용한다.
              // 주어진 위치 (row, col) 의 cell 에 주어진 값 value 를 넣는다.
              this._cells [row][col] = value ;
        }

        ......
```

요약

확인하자

- 다음의 내용을 잘 이해했는지 확인하자.
  - Model-View-Controller 설계 방식
  - 변수의 Scope
  - while 문과 for 문의 동등성
  - Static Class
  - Enum Class
  - 2차원 배열

❑ 생각해 볼 점

- Model-View-Controller 방식의 설계를 하는 이유는?
- Class "AppView" 의 모든 함수가 "static" 인 이유는?
- "public" 과 "private" 의 차이는?
- 상수 선언에 "static" 과 "final" 이 붙는 이유는?
- 슬라이드 27 쪽의, while 문과 for 문에서, 변수 "sum" 과 "count" 각각의 scope 는?

⇒ "생각해 볼 점" 에 대한 각자의 의견을 과제 보고서에 작성하시오.

보고서 작성과 제출

보고서 작성 방법

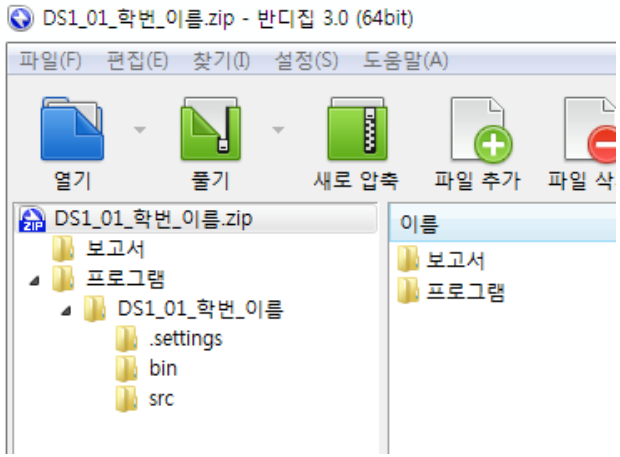
- 보고서는 PDF 파일로 만들어 제출한다.
- 겉장
  - 큰 제목: 자료구조 실습 보고서
  - 작은 제목: [제xx주] 숙제명 (예: [제01주] 마방진)
  - 제출일
  - 학번/이름

- 내용
  1. 프로그램 설명서
     1. 프로그램의 전체 설계 구조 (MVC 등)
     2. 함수 설명서 (주요 알고리즘 / 자료구조의 설명 포함)
     3. 종합 설명서
  2. 프로그램 장단점 / 특이점 분석
  3. 실행 결과 분석
     1. 입력과 출력 (화면 capture 하여 첨부)
     2. 결과 분석 (자신의 논리적 평가, 기타 느낀 점)
  4. "생각해 볼 점"에 대한 의견

     (슬라이드의 요약 부분에서 언급된 " 생각해 볼 점" 의 각 항목에 대해 자신의 의견을 서술)

과제 제출

- 파일 이름 작명 방법
  - DS01_학번_이름.zip
  - 폴더의 구성
    - DS01_학번_이름
      - 프로그램
        - 프로젝트 폴더 / 소스
        - 메인 클래스 이름 : _DS01_학번_이름.java
      - 보고서
        - 이곳에 보고서 문서 파일을 저장한다.
        - 입력과 실행 결과는 화면 image 로 보고서 문서에 포함시킨다.
        - 보고서는 pdf 파일로 만들어 제출한다.



[제 1 주 실습] 끝

많은 것을 얻는 한 학기가 되기를!!

Class Board [1]

```
public class Board {

        // Constant
        private static int EMPTY_CELL = -1 ;

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

        // Getters/Setters
        private int order() {
                return this._order;
        }
        private void setOrder (int newOrder) {
                this._order = newOrder;
        }

        private int[][] cells () {
                return this._cells ;
        }
        private void setCells (int[][] newCells) {
                this._cells = newCells;
        }

        public void setCellValue (CellLocation location, int value) {
                this.cells()[location.row()][location.col()] = value;
        }
        public int cellValue (CellLocation location) {
                return this.cells()[location.row()][location.col()] ;
        }

        ......
```

this.cells() ?

this.cells() == this._cells
this.cells()[row][col] == this._cells[row][col]