자료구조 실습 보고서

[제 01 주] 마방진

제출일	2017/3/10
학 번	201000287
소 속	일어일문학과
이 름	유 다훈

201000287 일어일문학과 유다훈

1 프로그램 설명서

1 주요 알고리즘 및 자료구조

- 알고리즘
  - 입력 : 3 보다 크거나 같고 99 보다 작거나 같으며 홀수 인 수
  - 출력 : 입력 x 입력 크기의 표에서 가로, 세로, 대각선의 합이 모두 같은 마방진
  -
  - 입력이 3 보다 작을 때는 차수가 너무 작다는 오류 출력
  - 입력이 99 보다 클 때는 차수가 너무 크다는 오류 출력
  - 입력이 짝수 일 때는 홀수이어야 한다는 오류 출력
  - 입력이 홀수이면 프로그램 실행
  - 입력된 차수가 N 일 때, 1 부터 N*N 까지의 정수를 한 번 씩 모두 사용하여 판을 채우되 가로, 세로, 대각선의 합이 모두 동일
  - 임의의 칸을 채운 후 다음으로 채워야 할 칸은 오른쪽 위로 올라간다.
  - 만일 오른쪽 윗칸이 이미 숫자로 채워져 있다면 현재 위치로부터 바로 한 칸 아래로 내려가 채우고, 다시 오른쪽 위로 올라가며 채워간다.
  - 만일 판의 끝에서 진행해야 한다면, 마방진이 동그란 원으로 붙어있다고 가정하고 대각선의 다음 위치로 이동한다.

- 자료구조
  - 차수 x 차수만큼의 마방진 판을 생성해야하는 가로세로 판의 역할을 할 2 차원 배열.
  - 차수가 너무 작은지, 너무 큰지, 홀수가 아닌지 검사를 할 때 사용할 값이 있는 enum
  - 차수의 유효한지 아닌지에 따라 에러메세지를 출력 여부를 결정할 값이 들어가 있는 enum

2 함수 설명서

Class	AppController			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	run()	없음	없음	마방진 프로그램을 실행시키는 메소드
	showOrderValidityErrorMessage(OrderValidity anOrderValidity)	차수의 유/무효성을 알려줌	없음	입력받은 차수가 무효할때 해당되는 에러메세지 출력
	showBoard(Board aBoard)	마방진 문제를 풀고 난 후 정답이 기록되어 있음	없음	마방진 문제의 답을 화면에 출력
	showTitleForColumnIndexes(int anOrder)	차수	없음	입력받은 차수의 -1 만큼의 타이틀컬럼을 출력
	showMessage(MessageID aMessageID)	에러메세지 혹은 알림	없음	에러메세지 혹은 알림에 따라 메세지를 출력하는 메소드

Class	AppView			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	Appview()	없음	없음	생성자 메소드
	int inputOrder()	없음	order	차수를 내부에서 입력받아 리턴하는 메소드
	void outputLine(String aString)	출력하고 싶은 문자열	없음	메세지 출력 및 줄바꿈 메소드.
	void output(String aString)	출력하고 싶은 문자열	없음	메세지 출력 메소드.
	void outputTitleWithOrder(int anOrder)	차수	없음	차수와 함께 문제풀이를 시작하는 것을 출력하는 메소드.
	void outputRowNumber(int aRouNumber)	가로 순서	없음	가로 출력. 0~차수-1 만큼.
	void outputCell(int aCellValue)	마방진 배열의 값	없음	특정 위치의 값을 출력

Class	Board			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	Board(int givenOrder)	차수	없음	주어진 차수만큼의 배열을 생성하고 초기값을 설정하는 생성자 메소드
	int order()	없음	차수	차수를 반환
	void setCell (CellLocation aLocation, int aCellValue)	마방진의 좌표와 해당 좌표에 넣을 값	없음	해당되는 위치에 값을 설정.
	int cell (CellLocation aLocation)	마방진의 좌표	마방진의 좌표에	파라미터로 받은 마방진의 좌표의 값을 반환

			해당하는 값	
	boolean cellsEmpty (CellLocation aLocation)	마방진의 좌표	참, 거짓	해당 좌표에 값이 있으면 거짓, 없으면 참을 반환

Class	CellLocation			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	CellLocation()	없음	없음	가로세로값이 각각 -1 인 생성자 메소드
	CellLocation(int givenRow, int givenCol)	설정할 가로, 세로값	없음	가로세로값을 주어지는 파라미터의 값으로 설정하는 생성자메소드
	void setRow(int newRow)	설정할 가로값	없음	가로위치 설정
	int row()	없음	가로값	가로위치 반영
	void setCol(int newCol)	설정할 세로값	없음	세로위치 설정
	int col()	없음	설정할 세로값	세로위치 반영

Class	MagicSquare			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	MagicSquare()	없음	없음	마방진을 만드는 기본 생성자 메소드
	MagicSquare(int givenMaxOrder)	만들어야할 최대 차수	없음	주어진 값만큼의 크기를 가진 마방진을 만드는 기본 생성자 메소드
	OrderValidity checkOrderValidity(int anOrder)	입력받은 차수	유/무효값	차수가 유효한지 검사하고 해당되는 유/무효값을 리턴하는 메소드
	Board solve(int anOrder)	입력받은 차수	마방진	주어진 차수에 따른 크기의 마방진을 생성하고 각 가로세로위치마다 값을 저장하는 메소드.

3 종합 설명서

- 매번 마방진 차수를 입력 받는다.
- 음수일 때는 마방진 프로그램을 종료한다.
- 음수가 아니면 차수의 오류 검사를 실행한다.
- 차수가 3 보다 작으면 3 보다 작다는 오류메세지를 출력한다.
- 차수가 99 보다 크면 차수가 너무 크다는 오류 메세지를 출력한다.
- 차수가 짝수 이면 차수가 짝수 라는 오류 메세지를 출력한다.
- 매번 입력된 차수에 대한 마방진 풀이 결과를 출력한다.
- 출력된 마방진은 차수 * 차수만큼의 크기를 가지며 가로, 세로, 대각선의 합이 모두 같은 판을 만들어 출력한다.

2 프로그램 장단점 분석

- 장점
  - 홀수의 차수를 입력 받아 차수 x 차수 크기만큼의 판에 가로세로대각선의 합이 모두 똑같은 마방진의 원리를 손쉽게 알 수 있다.
- 단점
  - 다음 단계의 숫자를 채울 때에 오른쪽 위로 상승한다는 법칙을 구현할 때에 마방진 판을 벗어나게 되면 마방진이 동그랗게 연결되어있다는 가정을 하고 해당되는 지점에 값을 넣어줘야하는 조건식을 작성해야한다.
  - 짝수의 수는 구현할 수 없다.

3 실행 결과 분석

1 입력과 출력

입력 - 입력된 차수가 무효한 경우

```
<<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): 101
오류 : 차수가 너무 큽니다. 99 보다 작거나 같아야 합니다.

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

입력 - 입력된 차수가 유효한 경우 3 일때

```
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
```

입력 - 입력된 차수가 유효한 경우 5 일때

```
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
```

입력 - 입력된 차수가 음수일 경우

```
마방진 차수를 입력하시오 (음수를 입력하면 종료합니다): -2

<<< 마방진 풀이를 종료합니다 >>>
```

2 결과 분석

- 3 보다 작은 수를 입력했을 때 : 차수가 너무 작다는 오류 메세지 출력
- 99 보다 큰 수를 입력했을 때 : 차수가 너무 크다는 오류 메세지 출력
- 짝수인 수를 입력했을 때 : 차수가 홀수이어야 한다는 오류 메세지 출력
- 홀수를 입력했을 때 : 입력된 차수 x 차수 만큼의 0 부터 시작하는 마방진 판을 생성하며 (0,차수/2)의 위치에서 1 부터 시작하여 오른쪽 위로 상승하여 차수 x 차수 만큼의 값을 채워나간다.
- 음수를 입력했을 때 : 마방진 프로그램을 종료한다.
  ⇒ 마방진의 알고리즘대로 정상적으로 작동하는 것을 확인할 수 있다.

4 소스코드

Class	AppController

```
public class AppController {
	private AppView _appView;
	private Board _board;
	private MagicSquare _magicSquare;

	public AppController() { //생성자.
		this._appView = new AppView();
		this._board = null;
		this._magicSquare = new MagicSquare();
	}

	public void run() {
		this.showMessage(MessageID.Notice_BeginMagicSquare); //마방진풀이를 시작합니다 라는
메세지 출력.
		OrderValidity currentOrderValidity; //차수가 유효한 홀수인지 검사하기 위한 enum.

		int order = this._appView.inputOrder();

		while(order > 0) { //만약 차수가 0 이상이라면.
			currentOrderValidity = this._magicSquare.checkOrderValidity(order); //입력받은
차수가 유효한지 검사한 값을 저장.

			if(currentOrderValidity == OrderValidity.Valid) { //만일 입력받은 차수가 유효한
값이라면,
				this._appView.outputTitleWithOrder(order); // 차수를 출력.
				this._board = this._magicSquare.solve(order); // 문제를 풀고,
				this.showBoard(this._board); //푼 문제를 출력한다.
			}
			else { //만약 차수가 유효한 값이 아니라면,
			this.showOrderValidityErrorMessage(currentOrderValidity); //에러메세지 출력.
		}

		order = this._appView.inputOrder(); //다음 문제를 풀기 위해 안내문을 출력하고 값을 다시
받기위한 준비를 한다.
	}
		this.showMessage(MessageID.Notice_EndMagicSquare); //차수가 0 이하라면,
		System.exit(0); // 프로그램을 종료한다.
	}

	private void showOrderValidityErrorMessage(OrderValidity anOrderValidity) { //차수가 유효한 값이
아닐때 출력하는 에러메소드.
		// TODO Auto-generated method stub
		switch (anOrderValidity) {
		case TooSmall : //차수가 너무 작을때,
			this.showMessage(MessageID.Error_OrderIsTooSmall); //차수가 너무 작다는
에러메세지 출력.
```

```
                                        break;

                                case TooLarge : //차수가 너무 클 때,
                                        this.showMessage(MessageID.Error_OrderIsTooLarge);  //차수가  너무  크다는
에러메세지 출력.
                                        break;

                                case NotOddNumber: //차수가 홀수가 아닐 때,
                                        this.showMessage(MessageID.Error_OrderIsNotOddNumber);  //차수가  홀수가
아니라는 에러메세지 출력.
                                        break;
                                        default:
                                                break;
                                }

                        }

                        private void showBoard(Board aBoard) { //문제를 풀고 답을 출력하는 메소드
                                // TODO Auto-generated method stub
                                CellLocation currentLoc = new CellLocation();
                                this.showTitleForColumnIndexes(aBoard.order()); //헤드컬럼을 출력하는 메소드.

                                for (int row= 0; row < aBoard.order(); row++) { //가로
                                        this._appView.outputRowNumber(row);
                                        for(int col = 0; col<aBoard.order(); col++) { //세로.
                                                currentLoc.setRow(row); //현재 배열에 가로세로 값을 지정해줌.
                                                currentLoc.setCol(col);
                                                this._appView.outputCell(aBoard.cell(currentLoc));  //좌표가  저장된
현재위의 값을 출력.
                                        }
                                        this._appView.outputLine("");
                                }
                        }
                        private void showTitleForColumnIndexes(int anOrder) { //헤드컬럼 출력.
                                // TODO Auto-generated method stub
                                this._appView.output("      ");//d

                                for (int col = 0; col < anOrder; col++) {
                                        this._appView.output(String.format("[%3d]",col));//헤드컬럼 출력
                                }
                                this._appView.outputLine("");

                        }
                        private void showMessage(MessageID aMessageID) { //메세지 출력메소드
                                // TODO Auto-generated method stub
                                switch (aMessageID) {
                                case Notice_BeginMagicSquare: //첫 프로그램이 시작되었을때.
                                        this._appView.outputLine("<<마방진 풀이를 시작합니다.>>");
                                        break;

                                case Notice_EndMagicSquare: //프로그램이 끝났을 때.
                                        this._appView.outputLine("");
                                        this._appView.outputLine("<<< 마방진 풀이를 종료합니다 >>>");
                                        break;

                                case Error_OrderIsTooSmall: // 차수가 너무 작을때.
                                        this._appView.outputLine("오류 : 차수가 너무 작습니다. 3 보다 크거나 같아야
합니다.");
```

```
                        break;
                case Error_OrderIsTooLarge://차수가 너무 클 때.
                        this._appView.outputLine("오류 : 차수가 너무 큽니다. 99 보다 작거나 같아야
합니다.");
                        break;
                case Error_OrderIsNotOddNumber: //차수가 홀수가 아닐 때.ru
                        this._appView.outputLine("오류 : 차수가 짝수입니다 홀수이어야 합니다..");
                        break;
                }

        }

}
```

Class	AppView

```
import java.util.Scanner;

public class AppView { //표시를 담당하는 클래스.
        private Scanner _scanner; //차수를 입력받는 스캐너.

        public AppView() { //생성자.
                this._scanner = new Scanner(System.in); //스캐너 선언.
        }

        public int inputOrder() { //차수를 입력받아 리턴하는 메소드.
                // TODO Auto-generated method stub
                this.outputLine("");
                this.output("마방진 차수를 입력하시오 (음수를 입력하면 종료합니다): ");

                int order = _scanner.nextInt(); //차수를 입력받아 저장하고 리턴.

                return  order;
        }

        public void outputLine(String aString) { //메세지 출력 및 줄바꿈 메소드.
                // TODO Auto-generated method stub
                System.out.println(aString);
        }

        public void output(String aString) { //메세지 출력 메소드.
                // TODO Auto-generated method stub
                System.out.print(aString);
        }
        public void outputTitleWithOrder(int anOrder) { //차수와 함께 문제풀이를 시작하는 것을 알리는 메소드.
                System.out.println("Magic Square Board : Order " + anOrder);
        }

        public void outputRowNumber(int aRowNumber) { //가로수 출력.
                System.out.printf("[%3d]", aRowNumber);
        }

        public void outputCell (int aCellValue) { //마방진의 내용을 출력하는 메소드.
                System.out.printf("  %3d", aCellValue);
        }
```

```
}
```

Class	Board

```
public class Board {
	private static int EMPTY_CELL = -1; //빈칸을 표현하기 위한 -1.
	private int _order; //차수.
	private int[][] _cell; //2 차원배열로 표현된셀.

	public Board(int givenOrder) { //생성자.
		this._order = givenOrder; //차수 설정.
		this._cell = new int [givenOrder][givenOrder]; // 차수 x 차수 크기만큼의 배열생성.

		for (int row = 0; row < givenOrder; row++) {
			for (int col = 0; col < givenOrder; col++) {
				this._cell[row][col] = Board.EMPTY_CELL; //배열에 초기값(빈칸) 설정.
			}
		}
	}


	public int order() {
		// TODO Auto-generated method stub
		return _order; //차수 반환.
	}
	public void setCell (CellLocation aLocation, int aCellValue) { //해당되는 위치에 값을 설정.
		this._cell[aLocation.row()][aLocation.col()] = aCellValue;
	}
	public int cell (CellLocation aLocation) { //해당되는 위치의 값을 반환.
		return  this._cell[aLocation.row()][aLocation.col()];


	}
	public boolean cellsEmpty (CellLocation aLocation) { //위치에 값이 있는지 없는지 확인.
		if (this._cell[aLocation.row()][aLocation.col()] != -1)  //만일 값이 빈칸이 아니라면,
			return false; //false 반환.
		else
			return true; //빈칸이라면 true 반환.

	}

}
```

Class	CellLocation

```
public class CellLocation { //셀의 위치를 가로세로로 표현할 수 있음.
	private int _row;
	private int _col;

	public CellLocation() { //객체를 생성한다.
		this._row = -1;
		this._col = -1;
	}
	public CellLocation(int givenRow, int givenCol) { //객체를 주어진 값으로 생성한다.
		this._row = givenRow;
		this._col = givenCol;
	}

	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;
	}

}
```

Class	MagicSquare

```
public class MagicSquare {
	private static int DEFAULT_MAX_ORDER = 99; //마방진을 구현할 수 있는 최대 차수값.

	private int _maxOrder;
	private int _order;
	private Board _board;

	public int maxOrder() {
		return this._maxOrder;

	}
	public int order() {
		return this._order;
	}

	public MagicSquare() { //기본 생성자.
		this._maxOrder = MagicSquare.DEFAULT_MAX_ORDER;
		this._order = 3;
		this._board = null;
	}

	public MagicSquare(int givenMaxOrder) { //차수가 주어졌을 때 생성자.
		this._maxOrder = givenMaxOrder;
		this._order = 3;
		this._board = null;

	}

	public OrderValidity checkOrderValidity(int anOrder) { //차수가 유효한지 검사하고 해당되는 유/무효값을
	리턴.
		// TODO Auto-generated method stub
		if (anOrder < 3) { //3 보다 작으면,
			return OrderValidity.TooSmall; //매우 작다 값.
		} else if (anOrder > 99) { // 99 보다 크면,
			return OrderValidity.TooLarge; //매우 크다 값.
		} else if (anOrder % 2 == 0) { // 차수를 나눠서 0 으로 떨어지면 짝수이므로,
			return OrderValidity.NotOddNumber; // 홀수가 아니다 값.
		} else
		return OrderValidity.Valid; // 모든 조건을 검사하고 통과하면 유효값 리턴.

	}
	public Board solve(int anOrder) { //문제를 푸는 메소드.
		// TODO Auto-generated method stub
		this._order = anOrder; //차수를 저장. 원래 기본값은 3 이었으나 바꿈.
		if(this.checkOrderValidity(anOrder) != OrderValidity.Valid) { //만일 차수가 유효하지 않으면,
			return null; //실행하지 않음.
		} else { //차수가 유효값이라면,
```

```
                this._board = new Board(this._order); // 차수 x 차수만큼의 판 객체 생성.

                CellLocation currentLoc = new CellLocation(0, this._order/2); //현재위치 객체 생성. 위치는 가로 0/ 세로 = 차수를 2 로나눈 가운데 값.
                CellLocation nextLoc = new CellLocation(); //아무 값도 없는 다음위치 객체 생성.

                this._board.setCell(currentLoc, 1);  // 보드의 첫번째 값을 설정. 현재위치에 1 입력.
                int lastValue = this._order * this._order; //보드 내부에 채워지는 값 중 최대값 설정. 차수 x 차수

                for(int cellValue = 2; cellValue <= lastValue; cellValue++){ //다음 값 2 부터 최대값까지 실행.
                        //다음 위치는 현재 위치로부터 오른쪽 위.
                        nextLoc.setRow(currentLoc.row()-1);  //다음 가로위치는 현재 가로위치로 부터 한 칸 위.
                        if(nextLoc.row() < 0) { //만약 0 번째줄보다 값이 낮아진다면, 마방진이 원통형으로 있다는 가정하에 바로아래값으로 가야하므로,
                        nextLoc.setRow(this.order()-1); //최대값보다 1 낮은 마방진의 제일 아래쪽으로 이동.
                        }
                        nextLoc.setCol(currentLoc.col()+1);  // 다음 세로 위치는 현재 세로위치로부터 한 칸 옆. 오른쪽으로 증가.
                        if(nextLoc.col() >= this.order()) { // 만약 최대값보다 크게된다면, 마방진이 원통형으로 있다는 가정하에 바로 왼쪽값(제일 낮은값)으로 .
                        nextLoc.setCol(0); //제일 낮은 값 0 으로 설정.
                        }

                        if (! this._board.cellsEmpty(nextLoc)) { //만약 다음위치에 이미 값이 들어있으면,
                        nextLoc.setRow(currentLoc.row()+1);  //현재위치로부터 바로 아래칸으로 내려간다.
                        nextLoc.setCol(currentLoc.col()); // 세로위치는 그대로.
                        }

                        //다음위치에 설정해놓은 가로세로값을 현재위치로 바꾸어 오른쪽 위로 진행한다.
                        currentLoc.setRow(nextLoc.row());
                        currentLoc.setCol(nextLoc.col());
                        // 현재위치에 들어있는 값을 그린다.
                        this._board.setCell(currentLoc, cellValue);

                }
                return this._board; //이렇게 그려진 값을 리턴한다.
        }


    }

}
```