자료구조 실습 보고서

[제01주] 마방진

제출일 : 2015-03-05

201402395 이승희

1. 프로그램 설명서

1) 주요 알고리즘 / 자료구조 / 기타

: 마방진이란 정사각형의 가로, 세로, 대각선의 합이 모두 동일하게 채워진 판으로, 한 변의 크기를 마방진 문제의 차수(order)라 하며 차수는 3보다 크거나 같은 홀수이다. 이 마방진을 MVC(Model-View-Controller) 디자인 패턴으로 프로그래밍하며 Controller, MagicSquare, View의 역할을 각각 AppController, MagicSquare, AppView 클래스가 맡게 된다.

2) 함수 설명서

(1) 출력 관련 함수들

Public void outputTitleWithOrder (int anOrder) {…}

Public void outputRowNumber (int aRowNumber) {…}

Public void outputCell (int anElement) {…}

Public void outputNextLine() {…}

Public void outputMessage (String aMessageString) {…}

(2) 입력 관련 함수들

Public int inputOrder() {…}

3) 종합 설명서

: 마방진을 계산하는 프로그램을 Model – View – Controller의 세 단계로 나누어 보면, Model에서 마방진을 계산 및 제공하여 Controller로 보내게 되고, Controller에서 입출력 지시를 내려 결과를 받아오면 View에서 키보드 입력 및 화면 출력하게 된다.

2. 프로그램 장단점 분석

: 클래스를 여러 개로 나누어 역할 분담을 확실하게 했다는 것이 장점이자 단점인 것 같다. 접근하기 쉽다는 것이 그 장점이고 익숙하지 않아 어렵게 느껴지는 것이 단점이다.

3. 실행 결과 분석

1) 입력과 출력

: 다음은 입출력을 관장하는 AppView 클래스의 소스코드이다.

```java
import java.util.Scanner;

public class AppView {

  private Scanner _scanner;

  public AppView() { //입력받을 객체를 생성한다.
    this._scanner = new Scanner(System.in);
  }

  public int inputOrder() { //사용자로부터 차수를 입력받는다.
    System.out.println("마방진 차수를 입력하시오 (음수를 입력하면 종료합니다.) : ");
    return this._scanner.nextInt();
  }

  public void outputTitleWithOrder(int anOrder) { //입력받은 차수를 출력한다.
    System.out.println("Magic Square Board: Order " + anOrder);
  }

  public void outputColNumber(int aColNumber) { //차수만큼의 행을 순서대로 출력한다.
    System.out.printf("[%3d]", aColNumber);
  }

  public void outputRowNumber(int aRowNumber) { //차수만큼의 열을 순서대로 출력한다.
    System.out.printf("[%3d]", aRowNumber);
  }

  public void outputCell(int anElement) { //성분들을 출력한다.
    System.out.printf("%5d", anElement);
  }
  public void outputLineNext() {
    System.out.println();
  }

  public void outputMessage(String aMessageString) { //마방진 계산이 불가능한 경우에 따른 메시지를 출력한다.
    System.out.println(aMessageString);
  }


}
```

2) 결과 분석

첫째로, 차수가 3과 5인 경우와 음수를 입력했을 때 종료되는 경우이다.

```
〈〈〈마방진 풀이를 시작합니다〉〉〉

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.) : 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.) : -1
〈마방진 풀이를 종료합니다〉
```

다음은 3보다 작은 수, 99보다 큰 수, 짝수인 경우이다.

```
〈〈〈마방진 풀이를 시작합니다〉〉〉

마방진 차수를 입력하시오 (음수를 입력하면 종료합니다.) : 1
오류 : 차수가 너무 작습니다. 3보다 크거나 같아야 합니다.

마방진 차수를 입력하시오 (음수를 입력하면 종료합니다.) : 180
오류 : 차수가 너무 큽니다. 99보다 작거나 같아야 합니다.

마방진 차수를 입력하시오 (음수를 입력하면 종료합니다.) : 4
오류 : 차수가 짝수입니다. 홀수이어야 합니다.

마방진 차수를 입력하시오 (음수를 입력하면 종료합니다.) : -1
〈마방진 풀이를 종료합니다〉
```

4. 소스코드

1) AppController

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

  public AppController() {
    this._appView = new AppView();
    this._magicSquare = new MagicSquare();
  }

  public void run() {
    this.showMessage(MessageID.Notice_BeginMagicSquare);
    OrderValidity currentOrderValidity;

    int order = this._appView.inputOrder();

    while (order > 0) {
      currentOrderValidity = this._magicSquare.checkOrderValidity(order);

      if (currentOrderValidity == OrderValidity.Valid) {
        this._appView.outputTitleWithOrder(order);
        this._board = this._magicSquare.solve(order);
        this.showBoard(this._board);
      } else
        this.showOrderValidityErrorMessage(currentOrderValidity);

      order = this._appView.inputOrder();

    }
    this.showMessage(MessageID.Notice_EndMagicSquare);
  }
private void showOrderValidityErrorMessage(OrderValidity anOrderValidity) {
  switch (anOrderValidity) {
  case TooSmall:
    this.showMessage(MessageID.Error_OrderIsTooSmall);
    break;
  case TooLarge:
    this.showMessage(MessageID.Error_OrderIsTooLarge);
    break;
  case NotOddNumber:
    this.showMessage(MessageID.Error_OrderIsNotOddNumber);
    break;
  default:
    break;
  }
}

private void showMessage(MessageID aMessageID) {
  switch (aMessageID) {
  case Notice_BeginMagicSquare:
    this._appView.outputMessage("〈〈〈마방진 풀이를 시작합니다〉〉〉₩n");
    break;
  case Notice_EndMagicSquare:
    this._appView.outputMessage("〈마방진 풀이를 종료합니다〉₩n");
    break;
  case Error_OrderIsTooSmall:
    this._appView.outputMessage("오류 : 차수가 너무 작습니다. 3보다 크거나 같아야 합니다.₩n");
    break;
  case Error_OrderIsTooLarge:
    this._appView.outputMessage("오류 : 차수가 너무 큽니다. 99보다 작거나 같아야 합니다.₩n");
    break;
```

```
    case Error_OrderIsNotOddNumber:
      this._appView.outputMessage("오류 : 차수가 짝수입니다. 홀수이어야 합니다.₩n");
      break;
    default:
      break;
    }

  }

  private void showBoard(Board aBoard) {
    CellLocation currentLoc = new CellLocation();
    System.out.print("    ");
    for (int i = 0; i < aBoard.order(); i++) {
      this._appView.outputColNumber(i);
    }
    System.out.println();
    for (int i = 0; i < aBoard.order(); i++) {
      this._appView.outputRowNumber(i);
      for (int j = 0; j < aBoard.order(); j++) {
        currentLoc.setRow(i);
        currentLoc.setCol(j);
        this._appView.outputCell(aBoard.cell(currentLoc));
      }
      this._appView.outputLineNext();

    }
  }

}
```

2) Board

```
public class Board {
  private static final int EMPTY_CELL = -1;

  private int _order;
  private int[] [] _cell;

  public Board(int givenOrder) {
    this._order = givenOrder;
    this._cell = new int[givenOrder] [givenOrder];
    for (int row = 0; row < givenOrder; row++)
      for (int col = 0; col < givenOrder; col++)
        this._cell[row] [col] = Board.EMPTY_CELL;
  }

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

  public void setCell(CellLocation aLocation, int aNumber) {
    this._cell[aLocation.row()] [aLocation.col()] = aNumber;
  }

  public int cell(CellLocation aLocation) {
    return this._cell[aLocation.row()] [aLocation.col()];
  }

  public boolean cellsEmpty(CellLocation aLocation) {
    if (this._cell[aLocation.row()] [aLocation.col()] == Board.EMPTY_CELL)
      return true;
    else
      return false;
  }
```

3) CellLocation

```
public class CellLocation {
  private int _row;
  private int _col;

  public CellLocation() {
    this._row = -1;
    this._col = -1;
  }

  public CellLocation(int givenRow, int givenCol) {
    this._row = givenRow;
    this._col = givenCol;
  }

  public void setRow(int aRow) {
    this._row = aRow;
  }

  public int row() {
    return this._row;
  }

  public void setCol(int aCol) {
    this._col = aCol;
  }

  public int col() {
    return this._col;
  }
}
```

4) MagicSquare

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

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

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

  public MagicSquare(int givenMaxOrder) {
    this._maxOrder = givenMaxOrder;
    this._order = 3;
    this._board = null;
  }

  public OrderValidity checkOrderValidity(int order) {
    if (order < 3)
      return OrderValidity.TooSmall;
    else if (order > 99)
      return OrderValidity.TooLarge;
    else if (order % 2 == 0)
      return OrderValidity.NotOddNumber;
    else
      return OrderValidity.Valid;
  }
```

```
public Board solve(int anOrder) {
  this._order = anOrder;
  if (this.checkOrderValidity(anOrder) != OrderValidity.Valid) {
    return null;
  } else {
    this._board = new Board(this._order);
    CellLocation currentLoc = new CellLocation(0, this._order / 2);
    CellLocation nextLoc = new CellLocation();

    this._board.setCell(currentLoc, 1);

    int lastNumber = this._order * this._order;
    for (int number = 2; number <= lastNumber; number++) {
      if (currentLoc.col() == _order - 1)
        nextLoc.setCol(0);
      else
        nextLoc.setCol(currentLoc.col() + 1);
      if (currentLoc.row() == 0)
        nextLoc.setRow(_order - 1);
      else
        nextLoc.setRow(currentLoc.row() - 1);
      if (!this._board.cellIsEmpty(nextLoc)) {
        nextLoc.setCol(currentLoc.col());
        if (currentLoc.row() == _order - 1)
          nextLoc.setRow(0);
        else
          nextLoc.setRow(currentLoc.row() + 1);
      }
      currentLoc.setRow(nextLoc.row());
      currentLoc.setCol(nextLoc.col());


        this._board.setCell(currentLoc, number);
      }
      return this._board;
    }
  }

  public int maxOrder() {
    return _order ^ 2;
  }

  public int order() {
    return _order;
  }

}
```

5) Main

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

6) MessageID

```
public enum MessageID {
  // Message IDs for Notices:
  Notice_BeginMagicSquare, Notice_EndMagicSquare,
  // messageIDs for Errors:
  Error_OrderIsTooSmall, Error_OrderIsTooLarge, Error_OrderIsNotOddNumber,
}
```

7) OrderValidity

```
public enum OrderValidity {
    Valid,
    TooSmall,
    TooLarge,
    NotOddNumber,
}
```

8) AppView 클래스의 소스코드는 위 3-1)에 첨부되어 있으니 따로 첨부하지 않았다.