실습 5

조건문, 순환문 및 Java 배열

목 적

- Java 조건문 및 순환문 이해
- 조건문에서의 논리적 에러에 대한 이해
- 기본 자료형을 원소로 갖는 배열의 선언 및 처리에 대한 이해

제어구조

1. 다음 프로그램을 `Control.java` 파일에 저장한다.

```
import java.util.*;

// An exploration of basic control structures.
class Control {

   // Processes input using a loop and a conditional.
   public static void main (String[] args) {

      Scanner in = new Scanner(System.in);

      System.out.println ("Enter a number: ");
      int num = in.nextInt();

      while (num < 20) {
         if (num%2 == 0)
            System.out.println (num);
         num = num + 1;
      }

  } // method main

} // class Control
```

2. 프로그램을 컴파일하고 실행한다. 프롬프트에서 `10`을 입력한다. 무엇이 출력되는가?

3. 이 프로그램은 무엇을 하기 위한 것인가? 프로그램의 목적을 달성하는데 `if` 문의 조건이 어떻게 사용되었는가?

4. 프롬프트에서 `25`가 입력되었을 때 무엇이 프로그램에서 출력 되는지를 설명하라.

5. 프로그램이 무한 루프에 빠질 가능성에 대하여 설명하라.

6. 입력된 수와 같거나 크고 `50` 이하인 정수에서 `5`로 나누어지는 모든 수를 출력하도록 프로그램을 수정한다. 수정한 프로그램을 여러 번 시험한다.

7. `0`보다 작은 수가 입력되면 에러 메시지를 보이도록 프로그램을 수정하라. 프로그램은 입력된 수가 `0`보다 클 경우에만 실행을 계속하도록 한다. 프롬프트도 알맞게 변경하고, 수정된 코드가 적절히 들여쓰기(indentation) 되었는지 확인한다. 프로그램을 컴파일한 후 여러 입력 값을 주어 실행해 본다.

8. `Control.java`의 최종본을 제출한다.

최대 값과 최소 값 찾기

루프에서 실행되는 공통적인 작업 중 하나는 일련의 여러 값에서 최대 값과 최소값을 찾는 것이다. 아래의 `Temps.java`는 24시간 동안 순차적으로 시간별 온도 측정 결과를 읽는 프로그램이다. 이 프로그램에 최대 온도와 최소 온도를 찾는 코드를 추가한다. 다음 지시를 따른다.

1. 프로그램을 디렉토리에 저장한 후 내용을 살펴 본다. 일정한 회수만큼 반복하므로 `for` 루프를 사용한 것에 주목하라. 먼저 최대 값을 찾는 코드를 추가한다. 일반적으로 루프에서 처리되는 일련의 값으로부터 최대값을 찾기 위해 다음 두가지가 필요하다.

    - 현재까지 찾아낸 최대값을 저장할 변수. 이 변수는 루프 전에 초기화 되어야 한다. 초기화 방법은 두가지가 있다. 하나는 모든 가능한 입력 값보다 작은 값으로 초기화하는 것이며, 두번째 방법은 검사할 첫번째 값으로 초기화 하는 것이다. 어느 방법이든 첫째 값이 처리된 후에는 변수가 첫째 값을 저장하고 있어야 한다. 여기서는 최대 온도를 저장하기 위해 `maxTemp` 변수를 선언하고 `-1000`도로 초기화 한다.
    - 반복 실행할 때마다 최대값 변수를 갱신하여야 한다. 이 것은 현재 값과 최대값 변수를 비교하여 현재 값이 크면 변수의 값을 현재 값으로 갱신하면 된다. 프로그램의 루프에 입력 값과 `maxTemp` 값을 비교하여 입력된 값이 크면 `maxTemp`에 저장하는 `if` 문을 추가한다.

2. 루프뒤에 최대값을 출력하는 코드를 추가하고 프로그램을 시험한다. 시험할 때 최소한 다음 세가지 경우를 생각해야 한다: 첫째 수가 최대값인 경우, 마지막 수가 최대값인 경우, 중간에 있는 수가 최대값인 경우. `HOURS_PER_DAY` 변수를 24보다 작은 값으로 하면 입력 회수를 줄여 시험을 쉽게 할 수 있다.

3. 최대값과 더불어 이와 연관된 정보를 원하는 경우가 종종 있다. 예를 들어 시험의 최고점수를 찾는 경우 최고점수를 받은 학생의 이름을 알고 싶어 할 것이다. 여기서는 최대 온도 값과 더불어 최대 온도가 나타난 시간에 대해 알기를 원한다. 따라서 최대 온도가 나타났을 때, 즉 `maxTemp`가 갱신될 때 `hour` 변수에 있는 현재 시간을 저장할 새로운 변수가 필요하다. 이를 위해 `timeOfMax` 변수(`int` 타입)를 선언하고, 최대온도와 함께 최대온도가 나타난 시간을 저장할 수 있도록 `if` 문을 수정한다.

4. 최대온도와 더불어 최대온도가 나타난 시간도 출력하도록 코드를 추가한다.

5. 유사한 방법으로, 최소온도와 최소온도가 나타난 시간을 찾는 코드를 추가한다.

6. `Temps.java`의 최종본을 제출한다

```
// *********************************************************
//   Temps.java
//
//   This program reads in a sequence of hourly temperature
//   readings (beginning with midnight) and prints the maximum
//   temperature (along with the hour, on a 24-hour clock, it
//   occurred) and the minimum temperature (along with the hour
//   it occurred).
// *********************************************************

import java.util.*;

public class Temps
{
    public static void main(String[] args)
    {
      final int HOURS_PER_DAY = 24;

      int temp;   // a temperature reading

      // print program heading
      System.out.println();
      System.out.println("Temperature Readings for 24 Hour Period");
      System.out.println();

      Scanner in = new Scanner(System.in);

      for (int hour = 0; hour < HOURS_PER_DAY; hour++)
      {
          System.out.print("Enter the temperature reading at " + hour +
                           " hours: ");
          temp = in.nextInt();
      }

      // Print the results
    }
}
```

문자 개수 세기

`Count.java`는 문자열을 읽고 그 속에 있는 빈 문자(blank character)의 수를 세는 프로그램의 골격을 보이고 있으며, 현재 선언문과 초기화문 그리고 결과 출력문으로 구성되어 있다. 여기에 추가해야 할 것은 문자열의 문자를 하나씩 검사하여 빈 문자의 개수를 세는 루프이다(빈 문자의 개수를 셀 때 `countBlank` 변수를 사용할 것). 문자열에 있는 총 문자 개수는 `String` 클래스의 `length` 메쏘드를 통해 알 수 있기 때문에 `for` 루프를 사용한다.

1. `for` 루프를 프로그램에 추가한다. `for` 루프 내부에서 `String` 클래스의 `charAt` 메쏘드를 사용하여 문자열의 각 문자를 호출하도록 하고, 빈 문자의 개수는 `countBlank` 변수에 저장한다.

2. 여러 구문(phrase)을 데이터 값으로 입력하여 프로그램이 제대로 작동하는지 시험한다.

3. 빈 문자뿐 아니라 다른 문자가 나타나는 회수도 셀 수 있도록 프로그램을 수정한다. 시간 제약상 여기서는 대·소문자를 구분하지 않고 4개의 문자 a, e, s, t 에 대해서만 회수를 세도록 한다. 이 경우 4개의 변수가 더 필요하게 된다(이름은 `countA`, `countE` 등으로 할 것). 중첩된 `if` 문을 사용하여 프로그램할 수도 있으나, 9개 문자(a, A, e, E, s, S, t, T, 빈 문자)를 다뤄야 하므로 `if` 문 대신 `switch` 문을 사용한다. 아래 보이는 바와 같이 `switch` 문의 `case`는 `ch` 변수를 사용하여 선택한다.

   ```
   switch(ch)
   {
      case 'a':
      case 'A':  countA++;
                 break;

      case ....

   }
   ```

4. 모든 결과를 출력하도록 프린트문을 추가한다.

5. 사용자가 중단할 때까지 구문을 입력받아 문자의 개수를 출력하는 작업을 반복하도록 프로그램을 수정한다. 이렇게 하려면 현재의 루프를 둘러 싼 다른 루프가 필요하다. 사용자가 “quit”을 입력하지 않는한 구문을 계속 입력받을 수 있도록 현재 루프를 둘러 싼 while 루프를 추가한다. 구문을 입력하거나 중지하라는 메시지(예, “Enter a phrase or type quit: “)를 보여주도록 프롬프트를 수정한다. 사용자가 여러 구문의 첫번째 구문을 입력할 때 문자 개수를 세기 시작하여야 하기 때문에 문자 개수를 세기 위한 변수가 while 루프 안에서 초기화되어야 하는 것에 주의하라. 중첩된 루프 등 추가된 코드에 대해 가독성을 위한 들여쓰기 등이 제대로 되어 있는지 프로그램을 살펴 본다.

6. `Count.java`의 최종본을 제출한다.

```
// ********************************************************
//   Count.java
//
//   This program reads in strings (phrases) and counts the
//   number of blank characters and certain other letters
//   in the phrase.
// ********************************************************

import java.util.*;

public class Count
{
  public static void main(String[] args)
  {
      Scanner in = new Scanner(System.in); // standard input
      String phrase;    // a string of characters
      int countBlank;   // the number of blanks (spaces) in the phrase
      int length;       // the length of the phrase
      char ch;          // an individual character in the string

      // Print a program header
      System.out.println();
      System.out.println("Character Counter");
      System.out.println();

      // Read in a string and find its length
      System.out.print("Enter a sentence or phrase: ");
      phrase = in.nextLine();
      length = phrase.length();

      // Initialize counts
      countBlank = 0;

      // a for loop to go through the string character by character
      // and count the blank spaces

      // Print the results
      System.out.println();
      System.out.println("Number of blank spaces: " + countBlank);
      System.out.println();
    }
}
```

매직 스퀘어 (primitive 타입 원소를 갖는 배열)

매직 스퀘어(magic square)는 2차원 배열을 응용할 수 있는 재미있는 문제이다. 매직 스퀘어는 각 행의 합, 각 열의 합, 두 대각선의 각 합이 모두 같은 정사각행렬을 말한다. 매직 스퀘어는 오랜 시간 연구되어 왔으며 잘 알려진 몇 개의 매직 스퀘어가 있다. 본 실습에서는 정사각행렬이 매직 스퀘어인지를 결정하는 프로그램을 작성한다.

1. `Square.java`는 정사각행렬을 읽어 들여 그 행렬이 매직 스퀘어인지를 말해주는 클래스 골격을 보이고 있다. 클래스를 보면 주어진 크기의 정사각 행렬을 만드는 생성자, 행렬에 원소 값을 읽어 들이는 메쏘드, 정사각 행렬을 출력하는 메쏘드, 각 행의 합을 구하는 메쏘드, 각 열의 합을 구하는 메쏘드, 두 대각선의 각 합을 구하는 메쏘드, 그리고 행렬이 매직 스퀘어인지 결정하는 메쏘드가 있다. 원소의 값을 읽어 들이는 메쏘드는 주어져 있다. 나머지 다른 메쏘드를 완성하라.

2. 주석문의 설명을 보고 나머지 코드를 완성해 나간다. 완성된 프로그램을 magicData 파일의 데이터를 입력받아 시험한다(프로그램은 표준 입력장치로부터 입력 받으므로 파일로부터 입력받을 수 있도록 redirection 하거나, "`Scanner in = new Scanner(new File("magicData"));`" 명령문을 사용). magicData 파일에 있는 처음 세개의 행렬은 매직 스퀘어로, 나머지 4개는 매직 스퀘어가 아닌 결과를 주어야 한다. magicData 파일의 맨 끝에 있는 -1은 입력 데이터가 더 이상 없음을 나타내기 위한 수이다.

3. `readSquare` 메쏘드에서 `for` 순환문 대신 "for-each" 순환문을 사용하여 원소의 값을 읽어 들이도록 프로그램을 수정하고 컴파일 한 후 시험한다.

4. `Square.java`의 최종본을 제출한다.

```
// ****************************************************************
// Square.java
//
// Define a Square class with methods to create and read in
// info for a square matrix and to compute the sum of a row,
// a col, either diagonal, and whether it is magic.
//
// ****************************************************************

public class Square
{

    int[][] square;
    Scanner in = new Scanner(System.in);

    //---------------------------------------
    //create new square of given size
    //---------------------------------------
    public Square(int size)
    {

    }

    //---------------------------------------
    //return the sum of the values in the given row
    //---------------------------------------
    public int sumRow(int row)
    {

    }

    //---------------------------------------
    //return the sum of the values in the given column
    //---------------------------------------
```

```
    public int sumCol(int col)
    {

    }

    //---------------------------------------
    //return the sum of the values in the main diagonal
    //---------------------------------------
    public int sumMainDiag()
    {

    }

    //---------------------------------------
    //return the sum of the values in the other ("reverse") diagonal
    //---------------------------------------
    public int sumOtherDiag()
    {

    }

    //---------------------------------------
    //return true if the square is magic (all rows, cols, and diags have
    //same sum), false otherwise
    //---------------------------------------
    public boolean magic()
    {

    }

    //---------------------------------------
    //read info into the square from the standard input.
    //---------------------------------------
    public void readSquare()
    {
    for (int row = 0; row < square.length; row++)
      for (int col = 0; col < square.length; col ++)
          square[row][col] = in.nextInt();
    }

    //---------------------------------------
    //print the contents of the square, neatly formatted
    //---------------------------------------
    public void printSquare()
    {

    }

}
```

```
// ********************************************************************
// SquareTest.java
//
// Uses the Square class to read in square data and tell if
// each square is magic.
//
// ********************************************************************

public class SquareTest
{
    public static void main(String[] args)
    {

      int count = 1;                 //count which square we're on
      Scanner in = new Scanner(System.in);
      int size = in.nextInt(); //size of next square

      //Expecting -1 at bottom of input file
      while (size != -1)
          {

            //create a new Square of the given size

            //call its read method to read the values of the square

            System.out.println("\n******** Square " + count + " ********");
            //print the square

            //print the sums of its rows

            //print the sums of its columns

            //print the sum of the main diagonal

            //print the sum of the other diagonal

            //determine and print whether it is a magic square

            //get size of next square
            size = in.nextInt();

          }
    }
}
```

magicData

```
3
8  1  6
3  5  7
4  9  2
7
30  39  48   1  10  19  28
38  47   7   9  18  27  29
46   6   8  17  26  35  37
5   14  16  25  34  36  45
13  15  24  33  42  44   4
21  23  32  41  43   3  12
22  31  40  49   2  11  20
4
48   9   6   39
27  18  21   36
15  30  33   24
12  45  42    3
3
6  2  7
1  5  3
2  9  4
4
3  16   2  13
6   9   7  12
10  5  11   8
15  4  14   1
5
17  24  15   8   1
23   5  16  14   7
 4   6  22  13  20
10  12   3  21  19
11  18   9   2  25
7
30  39  48   1  10  28  19
38  47   7   9  18  29  27
46   6   8  17  26  37  35
5   14  16  25  34  45  36
13  15  24  33  42   4  44
21  23  32  41  43  12   3
22  31  40  49   2  20  11
-1
```

순환문 변환

1. 순환문은 아래 열거한 것처럼 반복되는 특성에 더 적합한 순환문을 사용하는 것이 좋다.

   - `for` 순환문: 반복 회수가 알려진 경우에 적합
   - `While` 순환문: 반복 회수를 모르는 경우에 적합
   - `do-while` 순환문: 반복 회수를 모르며 최소한 한번은 순환문을 실행해야 하는 경우에 적합

   다음 `while` 순환문을 `do-while` 순환문으로 변환하라. 어떻게 수정하였는가?

```
public static void main(String[] args)
{
    Scanner in = new Scanner(System.in);
    int sum = 0;
    int n = 1;

    while (n != 0)
    {
        System.out.print("Please enter a number, 0 to quit: ");
        n = in.nextInt();
        if (n != 0)
        {
            sum = sum + n;
            System.out.println("Sum = " + sum);
        }
    }
}
```

2. 6 번에서 `while` 순환문을 `do-while` 순환문으로 변환하면 프로그램이 개선되는가? 그 이유는?

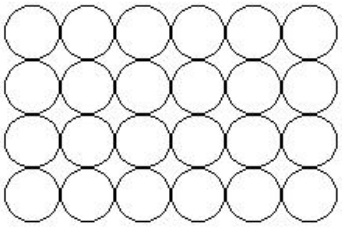
3. 다음 `while` 순환문을 `for` 순환문으로 변환하라. 어떻게 수정하였는가?

```
/**
  Program to compute the first integral power to which 2 can be
  raised that is greater than that multiple of a given integer.
*/
public static void main(String[] args)
{
  Scanner in = new Scanner(System.in);
  System.out.print("Please enter a number, 0 to quit: ");
  int n = in.nextInt();

  int i = 1;
  while (n * n > Math.pow(2,i))
  {
    i++;
  }

  System.out.println("2 raised to " + i
     + " is the first power of two greater than " + n + " squared");
  }
}
```

중첩된 순환문 (**Nested Loops)**

1. 24 개의 커피 캔을 가로 6 개, 세로 4 개로 배열하고 위에서 본 모양은 우측 그림과 같다.

   아래에 주어진 `CoffeeCansComponent` 클래스를 참조하여 옆 모양을 그려 주는 프로그램을 완성하고, 무엇을 수정하였는지 보고서에 기술하라.



```
// Draw a top view of 24 soda cans, that is 24 circles,
// arranged in a 4 x 6 grid.
import javax.swing.JComponent;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.Ellipse2D;

public class CoffeeCansComponent extends JComponent
{
   public void paintComponent(Graphics g)
   {
      Graphics2D g2 = (Graphics2D) g;
       …

      Ellipse2D.Double can = new Ellipse2D.Double(?, ?, ?, ?);
      g2.draw(can);

       …
    }
}
```

```
// Display a given figure.
import javax.swing.JFrame;

public class CoffeeCansViewer
{
   public static void main(String[] args)
   {
      JFrame frame = new JFrame();

      final int FRAME_WIDTH = 150;
      final int FRAME_HEIGHT = 140;

      frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      CoffeeCansComponent component = new CoffeeCansComponent();
      frame.add(component);

      frame.setVisible(true);
   }
}
```