Lecture 24

File Handling - 2

In last lecture we have discussed need and advantage of file handling in development of programs. File handling enhance scope of our programs and we can create programs having more functionality. We can create programs that can:

- take long inputs from files
- write output in files to share to other programs
- do tasks in parts where task done in any time can be saved
- fulfills needs of any business concern by saving and loading their running data [sales, purchases, employees, goods, debtors, creditors, bank cash etc. records]

We have also discussed reading and writing in files. Reading using ***Scanner*** class as similar to user input, writing using ***PrintWriter*** class similar to ***System.out.print*** or ***System.out.println*** functions we have used for output on screen in our programs. We have to import ***java.io.**** package for ***PrintWriter*** & ***File*** class. ***File*** class object is required in creation of ***Scanner*** class object. Interesting Scanner reads data from file instead of keyboard if file object is used in creation of ***Scanner*** class object. ***throws Exception*** phrase is to be added before starting curly brace of every function doing file handling code.

In this lecture we will start discussion on meta-data inside files. Meta data is extra information added into files by creator/ writer of file to help/ guide reader. We will discuss how to write meta-data and how to use it while reading before that we will discuss some examples of meta-data:

- count as meta-data about number of values to be read, stored in start of file
- for a table type data where each row have same number of columns, row & column count as meta-data, again stored in start of file
- for a table type data where each row may have different number of columns, row count as meta-data, stored in start of file, where column count as meta-data, stored before start of each row see data example:

```
4
3 23 32 16
4 19 15 28 34
2 23 39
3 15 18 19
```

Here 4 shows number of rows; whereas; 3,4,2,3 are number of columns in each row. Now we will discuss coding examples to handle meta-data.

Open a new file in textpad. (In eclipse File->New->File give name of file with extension, write data & save). Write following data and save file with name "cgpas.txt":

```
5 3.2 2.1 3.3 2.5 2.9
```

Here 5 is meta-data followed by 5 cgpas written with spaces as separator. Now we write code to read this file. First we will read count than declare double type array to read and store values:

```java
File file=new File("cgpas.txt");
Scanner in=new Scanner(file);
final int COUNT=in.nextInt();
double cgpas[]=new double[COUNT];
int i;
```

```java
double sum=0;
for (i=0;i<COUNT;i++){
      cgpas[i]=in.nextDouble();
      sum=sum+cgpas[i];
      ?(cgpas[i]+" ");
}
```

```
?ln("\n Average:"+sum/COUNT);
```

In this code after opening file we are using ***nextInt*** function to read count from file. Than we have declared double type array according to number of values stored in file. Later we are running loop according to size stored in ***COUNT*** and storing values in array and simultaneously taking sum as well. At the end of loop average is calculated and printed. Next you should play with this code by changing data file only and rerunning code. Change number of data values you may give more or lesser values change count stored as meta-data and rerun code without even compiling and see it will automatically work for any number of values provided meta-data is given correctly.

Next we are starting our example with real data. Below is the record of Pakistani player "Mohammad Hafeez" for T20 matches from 28-Aug 2006 to 28-Dec 2012, link is given below:

Scores	Wkts	Opposition	Start Date
46	-	England	**28-Aug-06**
25	-	South Africa	**2-Feb-07**
12	2	Kenya	**4-Sep-07**
18	1	Scotland	**12-Sep-07**
10	1	Sri Lanka	**17-Sep-07**
15	1	Australia	**18-Sep-07**
23	2	Bangladesh	**20-Sep-07**
32	0	New Zealand	**22-Sep-07**
1	0	India	**24-Sep-07**
DNB	1	Bangladesh	**1-May-10**
12	1	Australia	**2-May-10**
18	0	England	**6-May-10**
8	0	New Zealand	**8-May-10**
1	0	South Africa	**10-May-10**
DNB	0	Australia	**14-May-10**
14	0	England	**7-Sep-10**
13	1	South Africa	**26-Oct-10**
14	1	South Africa	**27-Oct-10**
24	1	New Zealand	**26-Dec-10**
46	-	New Zealand	**28-Dec-10**
34	-	New Zealand	**30-Dec-10**

Scores	Wkts	Opposition	Start Date
3	0	West Indies	**21-Apr-11**
71	4	Zimbabwe	**16-Sep-11**
51	3	Zimbabwe	**18-Sep-11**
13	1	Sri Lanka	**25-Nov-11**
25	2	Bangladesh	**29-Nov-11**
23	2	England	**23-Feb-12**
0	1	England	**25-Feb-12**
0	0	England	**27-Feb-12**
0	0	Sri Lanka	**1-Jun-12**
24	0	Sri Lanka	**3-Jun-12**
17	2	Australia	**5-Sep-12**
45	0	Australia	**7-Sep-12**
9	0	Australia	**10-Sep-12**
43	0	New Zealand	**23-Sep-12**
45	0	Bangladesh	**25-Sep-12**
15	2	South Africa	**28-Sep-12**
15	0	India	**30-Sep-12**
4	2	Australia	**2-Oct-12**
42	1	Sri Lanka	**4-Oct-12**
61	0	India	**25-Dec-12**
55	0	India	**28-Dec-12**

`http://stats.espncricinfo.com/ci/engine/player/41434.html?class=3;orderby=start;template=results;type=allround;view=match`

I have copied this data into MSExcel and saved as CSV (Comma Separated Values). Later I have opened this file and replace comma (,) with space and now I am writing code to read and display this data. I have faced another problem that is space between country names and while reading using ***next()*** to read String it was reading country names in two parts, therefore, I replaced space between country names with _ and finally very simple code to read and display data. Format can be made better by applying some checks on length of country name. Here is the code:

```
    File file=new File("../hafeez_t20.csv");
    Scanner in=new Scanner(file);
    System.out.println("Score\tWkts\tCountry\t\tDate");
    while (in.hasNext()){
        System.out.print(in.next()+"\t");
        System.out.print(in.next()+"\t");
        System.out.print(in.next()+"\t\t");
        System.out.println(in.next());
```

```
	}
```

I have written "../" in File name because I have stored file in outer folder where code was inside inner folder, if you have both files in same folder you should remove this and simply write file name as we did in previous code. Now this code is not storing any data just reading and displaying on the screen. If we want to store data for processing we cannot use array because we don't know the size of data, however, we may use Vector because Vector grows automatically with data addition, and also we don't need to mention size:

```
		File file=new File("../hafeez_t20.csv");
		Vector<String> scores=new Vector<String>();
		Vector<String> wkts=new Vector<String>();
		Vector<String> countries=new Vector<String>();
		Vector<String> matchDates=new Vector<String>();
		Scanner in=new Scanner(file);
		while (in.hasNext()){
			scores.add(in.next());
			wkts.add(in.next());
			countries.add(in.next());
			matchDates.add(in.next());
		}
		int i;
		System.out.println("Date\t\tCountry\tScore\tWkts");
		for (i=0;i<scores.size();i++){
			
		System.out.print(matchDates.get(i)+"\t"+countries.get(i)+"\t");
			if (countries.get(i).length()<8)
				System.out.print("\t");
			System.out.println(scores.get(i)+"\t"+wkts.get(i));
		}
```

In this code we are storing values in vector and later displaying them in different order and we are also doing formatting as well, which is visible in output. Now we are going to extend the code by writing data in another file in different format. That is writing scores and wkts only and for those matches for which scores and wkts are available. First of all we will count no of times batting done and no of times wkts taken:

```
		int i, noBat=0, noWkts=0;
		char ch1, ch2;
		for (i=0;i<scores.size();i++){
			ch1=scores.get(i).charAt(0);
			ch2=wkts.get(i).charAt(0);
			if (ch1>='0' && ch1<='9')
				noBat++;
			if (ch2>='0' && ch2<='9')
				noWkts++;
		}
		System.out.println("No of times bat:"+noBat);
		System.out.println("No of times wickets taken:"+noWkts);
```

We have counted because there are some matches where instead of score we have ***DNB*** because batsman have not taken turn, similarly where in case of wickets we have – hyphen where no bowling available. This is the reason why I have not used ***nextInt*** because ***nextInt*** can be used only if we have digits 0-9. For any other character ***InputMismatchException*** occur. Now we will write this data into another file where we will write only scores and wickets with meta-data. See the code:

```
	PrintWriter pw=new PrintWriter("../records.txt");
	pw.println(noBat);
	for (i=0;i<scores.size();i++){
		ch1=scores.get(i).charAt(0);
		if (ch1>='0' && ch1<='9')
			pw.print(scores.get(i)+" ");
	}
	pw.println("\n"+noWkts);
	for (i=0;i<wkts.size();i++){
		ch2=wkts.get(i).charAt(0);
		if (ch2>='0' && ch2<='9')
			pw.print(wkts.get(i)+" ");
	}
	pw.close();
```

In previous example we were reading data, here we are writing data. In two bold and italic lines we are writing meta-data; whereas; in loop we are writing actual data means scores and wickets separated by space. Last example is we have a file ***question.txt*** having 2 matrices. Matrix has table type of data of same type means rows and columns:

```
2 3
9 7 3
6 4 8
3 4
2 5 1 6
3 7 2 4
4 1 5 3
```

Here 2 3 means 2 rows and 3 columns followed by data of first matrix. Next 3 4 means 3 rows and 4 columns followed by data of second matrix. Here first we will see code this data into 2D arrays of type int:

```
	File file=new File("question.txt");
	Scanner in=new Scanner(file);
	int i, j, m1, n1, m2, n2;
	m1=in.nextInt();
	n1=in.nextInt();
	int mat1[][]=new int[m1][n1];
	for (i=0;i<m1;i++)
		for (j=0;j<n1;j++)
			mat1[i][j]=in.nextInt();
	m2=in.nextInt();
	n2=in.nextInt();
	int mat2[][]=new int[m2][n2];
	for (i=0;i<m2;i++)
		for (j=0;j<n2;j++)
			mat2[i][j]=in.nextInt();
```

Here once again we are reading data from file using meta-data. First we read m1, n1 than we declared 2D array of size m1 & n1 and read first matrix into 2D array mat1 using nested loop with conditions on m1 & n1. Than we read m2, n2, again declared 2D array of size m2 & n2 and read second matrix into 2D array mat2 using nested loop with condition on m2 & n2. Next student should write code to multiply these two matrices and store into third matrix of size m1 & n2. Yes if you recall matrix multiplication the resultant matrix has rows equal to number of rows of first matrix and columns equal to number of columns of second matrix. Here we are assuming that we have 2D array name ***result*** having result of multiplication of two matrices. Here is the code to write matrix ***result*** into file ***answer.txt***:

```
    PrintWriter pw=new PrintWriter("answer.txt");
    pw.println(m1+" "+n2);
    for (i=0;i<m1;i++)
         for (j=0;j<n2;j++)
              pw.print(result[i][j]+" ");
    pw.close();
```

Output written in file:

```
2 4
51 97 38 91
56 66 54 76
```

For practice student should read & write different type of data.