Chapter 10

File Management, data saving, data retrieving

Data File Creating:

Once we type something in computer, we need to store it in any storing media for future uses. Data can be in the form of text, audio, video or in any other formats. Different input instruments like Mouse, Microphone, camera, keyboard, scanner etc. helps to input data into the memory of the computer. Basic concept is that – all input devices produce analog signal of input data, it is then converted to digital signal, then coded as zero and one, 010110…., then as per coding converted to corresponding text which we see as data on the monitor of the computer. Data can be sound signal and if it is within the range of 2 to 20KHZ , then it is audio signal and our ear can hear that sounds. Camera produces Video data which is generally represented as unit MP(Mega Pixel). Overall, the concept is that it may be in any format either, text, audio or video, these are stored in a volatile memory or Semiconductor based memory (known as Chip). If power is off, then these data are lost. We need to store these data within permanent memory like Pendrive, Hard Disk, CD, DVD etc. for future reference.

Within a data file, not only the texts are stored but, data of other formats like- audio, video or picture files may be stored. In advanced database of Oracle, Foxpro, DB, all of these types of data can be stored. Data are nothing but some bits stored within the RAM of the computer, it has starting and ending address which we define as a particular data. So, a portion of a song may be stored as a data, another portion of a picture may be stored as picture data. But, these are advanced works of programming.

A basic diagram of read/write of data to storage media is shown below:

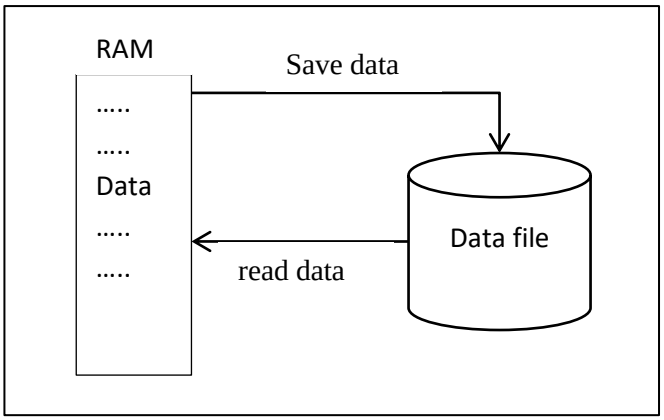


Fig. 10.1. a diagram shows basic concept of read/write data to permanent storage device.

Data are stored in a file giving a name. Within any software the 'File' menu do these operations like- 'Save-Open-Close-Delete etc.'. In text file, data are simply stored as per ASCII coded format. It is the simplest format of storing data. But, for other formats of file, data are encoded as per coding scheme of the format. A file can be of different types like: text, data, MSword, PDF, JPEG, WAV, MP4 formats etc. Using program, we store or save data in a file following some encoding rule for future uses; again we need to read those data, and display to the output devices. Output devices are: monitor, printer, harddisk, speaker, ECG printer etc. Data Management is another subject which deal about read, write, processing, security, reliability etc. of data. Here, in this chapter, simple data file management will be discussed through different programs.

Concept of Characters of Keyboard:

A data file contains characters which may be printable characters or non-printable characters. A file can be an empty file also having no data inside. When we input something, it is stored in a particular memory location of the RAM, and when we complete typing or mark as end of data, that is also noted as last memory address of the data. On the keyboard, there are various characters and each of the characters has a ASCII (American Standard Code For Information Interchange) code number value. When we press any character/symbol/function key, corresponding code value goes into the computer through the software. For easy understanding, ASCII value of capital letters A, B, C .. are 65, 66, 67 etc. and for small letters like: a, b, c, .. are 97, 98, 99 etc. for SPACE it is 32, for ESC it is 27 etc. A chart is given below mentioning all ASCII values.

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP
1	00000001	001	01	SOH	33	00100001	041	21	!
2	00000010	002	02	STX	34	00100010	042	22	"
3	00000011	003	03	ETX	35	00100011	043	23	#
4	00000100	004	04	EOT	36	00100100	044	24	$
5	00000101	005	05	ENQ	37	00100101	045	25	%
6	00000110	006	06	ACK	38	00100110	046	26	&
7	00000111	007	07	BEL	39	00100111	047	27	'
8	00001000	010	08	BS	40	00101000	050	28	(
9	00001001	011	09	HT	41	00101001	051	29	)
10	00001010	012	0A	LF	42	00101010	052	2A	*
11	00001011	013	0B	VT	43	00101011	053	2B	+
12	00001100	014	0C	FF	44	00101100	054	2C	,
13	00001101	015	0D	CR	45	00101101	055	2D	-
14	00001110	016	0E	SO	46	00101110	056	2E	.
15	00001111	017	0F	SI	47	00101111	057	2F	/
16	00010000	020	10	DLE	48	00110000	060	30	0
17	00010001	021	11	DC1	49	00110001	061	31	1
18	00010010	022	12	DC2	50	00110010	062	32	2
19	00010011	023	13	DC3	51	00110011	063	33	3
20	00010100	024	14	DC4	52	00110100	064	34	4
21	00010101	025	15	NAK	53	00110101	065	35	5
22	00010110	026	16	SYN	54	00110110	066	36	6
23	00010111	027	17	ETB	55	00110111	067	37	7
24	00011000	030	18	CAN	56	00111000	070	38	8
25	00011001	031	19	EM	57	00111001	071	39	9
26	00011010	032	1A	SUB	58	00111010	072	3A	:
27	00011011	033	1B	ESC	59	00111011	073	3B	;
28	00011100	034	1C	FS	60	00111100	074	3C	<
29	00011101	035	1D	GS	61	00111101	075	3D	=
30	00011110	036	1E	RS	62	00111110	076	3E	>
31	00011111	037	1F	US	63	00111111	077	3F	?

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
64	01000000	100	40	@	96	01100000	140	60	`
65	01000001	101	41	A	97	01100001	141	61	a
66	01000010	102	42	B	98	01100010	142	62	b
67	01000011	103	43	C	99	01100011	143	63	c
68	01000100	104	44	D	100	01100100	144	64	d
69	01000101	105	45	E	101	01100101	145	65	e
70	01000110	106	46	F	102	01100110	146	66	f
71	01000111	107	47	G	103	01100111	147	67	g
72	01001000	110	48	H	104	01101000	150	68	h
73	01001001	111	49	I	105	01101001	151	69	i
74	01001010	112	4A	J	106	01101010	152	6A	j
75	01001011	113	4B	K	107	01101011	153	6B	k
76	01001100	114	4C	L	108	01101100	154	6C	l
77	01001101	115	4D	M	109	01101101	155	6D	m
78	01001110	116	4E	N	110	01101110	156	6E	n
79	01001111	117	4F	O	111	01101111	157	6F	o
80	01010000	120	50	P	112	01110000	160	70	p
81	01010001	121	51	Q	113	01110001	161	71	q
82	01010010	122	52	R	114	01110010	162	72	r
83	01010011	123	53	S	115	01110011	163	73	s
84	01010100	124	54	T	116	01110100	164	74	t
85	01010101	125	55	U	117	01110101	165	75	u
86	01010110	126	56	V	118	01110110	166	76	v
87	01010111	127	57	W	119	01110111	167	77	w
88	01011000	130	58	X	120	01111000	170	78	x
89	01011001	131	59	Y	121	01111001	171	79	y
90	01011010	132	5A	Z	122	01111010	172	7A	z
91	01011011	133	5B	[	123	01111011	173	7B	{
92	01011100	134	5C	\	124	01111100	174	7C	|
93	01011101	135	5D	]	125	01111101	175	7D	}
94	01011110	136	5E	^	126	01111110	176	7E	~
95	01011111	137	5F	_	127	01111111	177	7F	DEL

As theoretical discussion done, to read a file from Hard Disk(HD), a indicator is required which points to the first character of the file residing within the HD or any storage device, reads characters one by one and puts within the temporary memory RAM. When characters are put within RAM, it also needs a file handler which can manage starting location of first character and the other locations of data within the RAM. In this way a planned system is developed to locate each of the characters. But, users can't see the inside process of the computer, only the system architect knows how they created the system internally.
The program shown below will read characters one by one and will print the Hexadecimal code of the character and the same character in ASCII code.

```
//RDemo.java: demonstration of reading data from a text file. To run this program a
text file //demofile.txt is required.
import java.io.*;
public class RDemo
{
        public static void main(String [] args) throws IOException
        {
          Reader  r = new FileReader("demofile.txt");
          int num;
          while ((num = r.read()) != -1)
          {
            //System.out.println(num +"\n");
            System.out.printf("ASCII code in Decimal is %d   of %<c \n", num);
        }
        r.close();
  }
}
```

OUTPUT:

```
C:\example>java RDemo
ASCII code in Hexa Decimal is 41   of "A"
ASCII code in  Decimal is 65   of "A"
ASCII code in Hexa Decimal is 42   of "B"
ASCII code in  Decimal is 66   of "B"
ASCII code in Hexa Decimal is 43   of "C"
ASCII code in  Decimal is 67   of "C"
ASCII code in Hexa Decimal is 44   of "D"
ASCII code in  Decimal is 68   of "D"
ASCII code in Hexa Decimal is 45   of "E"
ASCII code in  Decimal is 69   of "E"
ASCII code in Hexa Decimal is 46   of "F"
ASCII code in  Decimal is 70   of "F"
ASCII code in Hexa Decimal is 47   of "G"
ASCII code in  Decimal is 71   of "G"
```

ASCII code in Hexa Decimal is 48 of "H"
ASCII code in Decimal is 72 of "H"
ASCII code in Hexa Decimal is 49 of "I"
ASCII code in Decimal is 73 of "I"
ASCII code in Hexa Decimal is 4a of "J"
ASCII code in Decimal is 74 of "J"
…..
……

Discussion:
To run this program, we have to create a data file 'demofile.txt' and put "ABCDEFGHIabcdefghi" within this text file. This program will read each character separately from the data file 'demofile.txt' and then print the ASCII values of those characters. As example, we have put first ten capital and small alphabets; user may put any characters and check the value comparing with the value mentioned in the ASCII charts given above.

Extra work: it is suggested to change the print line as mentioned below, save and run the program, you will get only the ASCII numbers.
System.out.println(num +"\n");

If you look at the program, data was read from the data file using the method **r.read()** where 'r' is the object of the class **Reader.** This method reads individual characters from the data file and returns the corresponding ASCII value of that character. This theoretical concept is demonstrated from the line as mentioned below:
System.out.printf("ASCII code in Decimal is %d of %<c \n ", num);

If we want to print hexadecimal numbers, then, **%x** to be given instead of **%d**. Conversion of decimal number to hexadecimal number is as shown below:

$(65)_{10} = (41)_{16}$
$(66)_{10} = (42)_{16}$ etc.

Decimal number is divided by 16 to get hexadecimal number. As example, if 65 is divided by 16, quotient is 4 and remainder is 1, so (41) is the hexadecimal number.

Creating data file:

There are some differences between text file and data file. Text file is the simplest form of coding of digital bits to ASCII characters. ASCII text file can be visible by any text reader/editor. But, data file is the encoded form of text and can't be visible by other text or word editors. But the decoder of the same system can read the data. Advantages are that generally unauthorized person can't view contents of data. In the program below, a marks data file is created automatically to input 15 marks in a data file, JAVA encodes the system, so, it will not be possible to read data by any text editor. Because, text editor does not know the encoding scheme of conversion of data.

```
//Wdata2.java : When you write any data- should remember the type of data was
//saved.
// when read data from the same data file should read as the same data type.
import java.io.*;
class WData2
{
        public static void main(String[] args)
        {
          File in= new File("marksRND.dat");
          //DataInputStream din=null;
          DataOutputStream dout=null;
          try
          {
                    dout=new DataOutputStream(new FileOutputStream(in));
                    for(int i=0;i<15;i++)
                    {
                              dout.writeInt((int)(Math.random()*100));
                    }
          }
          catch( e)
          {
                    System.out.println(e.getMessage());
          }
           finally  {
              try {
                  dout.close();      }
              catch(IOException e){}
           }
        }
}
```

OUTPUT: nothing will be displayed but, it will create a new data file 'marksRND.dat' with 15 marks as encoded form. To see what are stored within the data file, if we try to display as:

```
E:\example>type marksRND.dat
O01^`↔7#G@RB3B▬
```

Discussions: When data file is tried to print, we see some characters are displayed but no numbers are printed. How can we be ensured that data were stored properly? As theory discussed before that text editor can't display encoded data. That is happened in this case. The DOS command 'type' can display contents of text data only, so, when tried, it displayed garbage output. The package 'java.io', known as input-output package, contains all classes and methods to handle input-output jobs. There are two classes 'DataInputStream' and 'DataOutputStream' which contain all necessary methods, constructors to handle data input-output jobs. This is known as

'read-write' of data. As this program is the example of data saving or writing to HD, only the class 'DataOutputStream' is used here. Which data are saved? If you look at the line shown below, it has two sections: one is writeInt() and the other is Math.random():

dout.writeInt((int)(Math.random()*100));

The random() method creates decimal numbers >0 and <1, i.e. it creates numbers between zero and one, but never equal to zero or equal to one. As example:

0.68 * 100 = 68

0.79 * 100 = 79

It is the logic why we multiplied by 100.

At end of the above program, Exception handler is used. Generally, for any input-output jobs, some common errors may happen as: keyboard, mouse, scanner etc. are damaged or output devices like: HD, CD, DVD, soundbox, printer etc. are damaged then exception error may be generated. Other exception errors are: data type mismatching, read-write abnormal errors etc. To manage these errors and to display information about the errors, try{ } and catch{ } error handlers are used. The class IOException contain all necessary information to manage exceptions.

```
//WriteData2.java: When you write any data- should remember the type of data.
// when read data from the same data file should read as the same data type.
import java.io.*;
import java.util.Scanner;
class WriteData2
{
   public static void main(String[] args)
  {
          File in= new File("marks2.dat");
          DataOutputStream dout=null;
         Scanner inp = new Scanner(System.in);
          int p;
          try
          {
              dout=new DataOutputStream(new FileOutputStream(in));
              for(int i=0;i<5;i++)
              {
                  System.out.print("\n Input Total Mark: ");
                  p = inp.nextInt();
                  dout.writeInt(p);
              }
           }
          catch(IOException e)
          {
             System.out.println(e.getMessage());
          }
          finally
          {
           try
          {
             dout.close();
```

```
        }
            catch(IOException e){}
    }
  }
}
```

OUTPUT:
Input Total Mark: 650

Input Total Mark: 765

Input Total Mark: 680

Input Total Mark: 820

Input Total Mark: 723

Discussion: when this program is run, it stores 5 numbers in the data file **marks2.dat**. After that for checking what it has stored within the data file, we see some non-readable characters are displayed. Program has done correct job, if we want to see data, we have to write another program for reading data from the data file. Java writes data as encrypted way having not to be read by any text editors. How the data is encrypted that logic remains within the compiler and user does not know that. Any file contains some header bytes of information of type of file, size, encryption, location of stored memory, user of file etc., then, next part is the data section stored as encrypted way.
As an example, we tried to see data by typing the name of the data file but, found some non-readable codes.

E:\example>type marks2.dat
☻è☻²☻¿♥4☻ ╚
E:\example>

Here, it is also proved that JAVA stored data in a file by encoding data to another form. When a file is created it contains header block where some information of the file are recorded. Generally, header block contains name of file, size of file, starting location in the storage device, encoding name or pattern etc. These information varies from software to software, as example, header block of MSWORD, notepad, MATLAB, ORACLE, C++ etc. are different.

Reading Data File:

Using program, if we like to read a file or open a file, we have to be confirmed that the file is existing in the storage device by name C:, D:, E: etc. So, name of the file with extension name should be mentioned. Extension name(.doc, .dat, .java, .class, .xls etc.) is the type of file. The main point to understand is that JAVA can read/open a file which is created by JAVA programming.
An important question is that- why can't particular software read files of other software? As example, MATLAB can't read ORACLE data file or MSWORD can't read MATLAB file etc. The reason behind is that the information stored within the header block of the file is known by that software only, so, it can't read data of other software.

Below, a program is shown to read a data file which was created by JAVA by our previous program. In previous program 'marks2.dat' data file was created and that is existing in the current location of the current drive of the computer. So, with the name of data file, the name of path is not required to mention. Because java searches the file in the default drive, we could mention as: 'C:\example\marks2.dat' also.

```
//ReadData2.java: When you write any data- it stores type of data also.
// When you read data from the same data file- should read as same data type.

import java.io.*;
class ReadData2
{
   public static void main(String[] args)
  {
     File in= new File("marks2.dat");
     DataInputStream din=null;
     //reading data from the data file
     try
    {
          din=new DataInputStream(new FileInputStream(in));
          for(int i=0;i<15;i++)
          {
             System.out.println(din.readInt());
          }
      }
      catch(IOException e)
     {
          System.out.println(e.getMessage());
     }
     finally
    {
          try
          {
            din.close();
          }
          catch(IOException e){}
     }
  }
}
```

OUTPUT:
650
765
680
820
723

Discussions: in this program, **din.readInt()** reads each integer numbers from the data file and displays on the screen. Five numbers were stored in the data file and giving a loop of 5 times, all five numbers are read. If more numbers are tried to read, it will give error message as **null** which indicates end of file. In the same way other types of data like: long, short, double, float, String etc can be stored and read also using the same logic. This is the basic concept of handing data file. There are some

other operations like, insert, delete, append and edit operations on data file and will be discussed later.

Home work: above two programs were written for integer types of data. You can edit those two programs for float or other type of numbers.

Read-Write operations:

In previous examples, single read or write operation was demonstrated in a single program. When we write or save some data, we need to know whether data are saved properly or not. Using text editor we can't see data file saved by JAVA program; we need to take help of reading program which can read data from data file. Both the jobs are demonstrated by the program shown below. First some data are saved and then those data are read and displayed.

```
//RWData3.java: both the write and read jobs are in a same program.
// if data is saved as integer, at the time of reading data it should be integer also.

Import  java.io.*;
class RWData3
{
        public static void main(String[] args)
        {
                File in= new File("marksRND.dat");
                DataInputStream din=null;
                DataOutputStream dout=null;
                try
                {
                        dout=new DataOutputStream(new FileOutputStream(in));
                        for(int i=0;i<5;i++)
                        {
                                dout.writeInt((int)(Math.random()*100));
                        }
                        dout.close();
                        // data storing into data file is complete.
                        //now read those data
                        din=new DataInputStream(new FileInputStream(in));
                        for(int i=0;i<5;i++)
                        {
                                System.out.printf(din.readInt());
                        }
                        din.close();
                }
                catch(IOException e)
                {
                        System.out.println(e.getMessage());
                }
        }
}
```

Discussion: In this program, first we have stored 5 numbers of random numbers generated by Random() method , stored it and closed the file handler 'dout'. Then

we started another file input handler ‘din’ for reading those numbers. As it is known that there are 5 numbers so we added another loop to read those 5 numbers one by one. Output of the program is:

```
E:\example>java RWData3
75
69
98
67
25
```

Home work: for input numbers, we used random number generators, instead of that, edit this program to input numbers from keyboard and save in the data file, read and display those numbers.

Text data read-write operation:

In previous examples, numerical data input-output operations are demonstrated. Generally data saving to hard disk is known as output operation and reading data from hard disk is known as input operation. In default drive, we have put a text file ‘data1.txt’ which will be read one by one for all characters and will be written to the second file ‘data2.txt’ one by one. There are two file handlers ‘Fi’ for input and ‘Fo’ for output respectively. The ‘read()’ method reads characters one by one and when reaches at end of the file then it returns ‘-1’ meaning nothing to be read. Till ‘-1’ is not faced, the read character from the file ‘data1.txt’ is written to ‘data2.txt’ file, in this way all data will be saved to the second data file and a new files will be created. It looks like copy the contents of a file to another one by another name.

```
//FileRW.java:  a program to work on File operation. In this program
// techniques of open, read, write and close a data file are demonstrated.

import java.io.*;
class FileRW
{
        public static void main(String[] args)
        {
                File Fi= new File("data1.txt");
                File Fo= new File("data2.txt");
                FileReader in1=null;
                FileWriter out1=null;
                try
                {
                        in1=new FileReader(Fi);
                        out1=new FileWriter(Fo);
                        int p;
                        while((p=in1.read()) != -1)
                        {
                                out1.write(p);
```

```
                }
            }
            catch(IOException e)
            {
                System.out.println(e);
                System.exit(-1);
            }
            finally
            {
                try
                {
                    in1.close();
                    out1.close();
                }
                catch(IOException e){}
            }
        }
}
```

OUTPUT:
Before running the program, we have put these contents within 'data1.txt' file and after running the program, the contents of 'data2.txt' file is displayed below. It is shown that all contents are copied within the 'data2.txt' file.

Contents of "data1.txt" source file is:
Democracy is the standard mode of governing the country. How Democracy will be practiced, played and exercised that depends on the leaders of the country. Now a days, most of the governing rules of the country are created by the members of the Parliament.

Contents of "data2.txt" file after running the program:
Democracy is the standard mode of governing the country. How Democracy will be practiced, played and exercised that depends on the leaders of the country. Now a days, most of the governing rules of the country are created by the members of the Parliament.

Discussions: Any file operation is controlled by file handler as:
`File Fi= new File("data1.txt");`
It has created an indicator 'Fi' to handle the exiting file 'data1.txt', but not mentioned whether this handler will be used for reading or writing purposes. For that 'in1' object is created with null data.
`FileReader in1=null;`
In that way, for writing purposes another two 'Fo' and 'out1' are created and the technique of using can be seen within the program shown above. Advanced operations of try{}, catch{}, IOException etc. are important for any input-output file operations.

File operation with Arrays:

Before, read-write operations with numerical and character types of data were discussed. Data contents may be stored within array and that one can be saved at a time resulting all data within the array will be saved within the data file. Advantage of using array in this case is that as array name is a single variable, read-write operation will be by name of array and all elements will be saved at a time.

```
//FileW.java: an array contains three names and saved in data file.
//all characters present in array are stored in a text file 'list.txt'.
import java.io.*;
class FileW
{
    public static void main(String[] args)
   {
        byte names[ ]={'G','A','N','D','H','I','\n','I','N','D','I','R','A','\n','M','O','D','I','\n'};
                FileOutputStream out1=null;
                try
                {
                        out1=new FileOutputStream("list.txt");
                        out1.write(names);
                        out1.close();
                }
                catch(IOException e)
                {
                        System.out.println(e);
                        System.exit(-1);
                }
        }
}
```

OUTPUT:
Using notepad or any text editor, you can see contents of text file
Contents of "list.txt" file is:
GANDHI
INDIRA
MODI

Discussions: in this program, careful observation to the contents of the array 'names', you will see, after each name a '\n' escape sequence for a new line is inserted. This is for new line and effect can be seen in output and three names in three different lines. As data will be stored (output) in text file, we created a file handler 'out1' and writes data as:

```
out1.write(names);
```

When saving of data is finished, we should close the file as:

```
out1.close();
```

It is a standard practice to close a file if opened once. Because, a file handler occupies some memory, if closed, that memory is released for other operations. If exception handler catches any unusual events, it will exit the JAVA virtual machine to the operation system. It means that program will exit unusually without returning any value. The statement

System.exit(-1);

will exit from java program but, not from operating system. If exit(n) where n is any integer number, then the program will terminate normally.

Now, another example shown below which will read a text file from the default drive and print on the screen. The read() method reads data till last character, when end of file is reached it returns '-1' and the loop of reading characters one by one stops. To run the program, a text file say 'list.txt' should be present in default drive.

```
//FileRead.java: a program to read all characters from a text File.
import java.io.*;
class FileRead
{
	public static void main(String[] args)
	{
		FileInputStream in1=null;
		int p;
		try
		{
			in1=new FileInputStream(args[0]);
			while((p=in1.read()) != -1)
			{
				System.out.print((char) p);
			}
			in1.close();
		}
		catch(IOException e)
		{
			System.out.println(e);
			System.exit(-1);
		}
	}
}
```

OUTPUT:

Compiling the program:

E:\example>javac FileRead.java

To run this program, we have to mention a data file name from which all data to be read. In this example, "list.txt" is the data file and to be given as the command line argument as shown below. Kindly, note that "list.txt" file was created by previous program and we assume that some text is available within that file.

Execute the program with argument zero as the name of text file:
E:\example>java FileRead list.txt
GANDHI
INDIRA
MODI
BIDEN

Discussion: if we look at the program, we will see there is a line

```
in1=new FileInputStream(args[0]);
```

where 'args[0]' is the name of the text file 'list.txt'.
What is this parameter argument? Our program is 'FileRead' and it takes 'list.txt' as first argument which is considered as the first input file to the program 'FileRead'.
Here, arg[0] is the "list.txt". As we want to read data from a text file, this job is known as input job meaning is that computer will input data to memory variable reading from the text file. So, it is known as input job; anyway if we fill computer memory that is known as input and from computer memory if we write or print to other devices(printer, HD, Soundbox, monitor) is known as output job.
Let us look at this statement:

```
FileInputStream in1=null;
```

Here, 'FileInputStream' is the class name available under the package 'io' and it reads streams of byte from the source. The object 'in1' is created and assigned with null data. The object is just created as a valid object and then it is made workable as:

```
in1=new FileInputStream(args[0]);
```

By this statement 'in1' is the object handler(or pointer) which locate the memory starting position from where the byte stream will be managed within the memory. Here, args[0] is the text filename provided in front of the program; if we put other names in front of the program as:

```
FileRead list.txt abc.txt xyz.txt
```

Then, arguments are like this: args[0] = list.txt, args[1]=abc.txt and agrs[2]=xyz.txt
Arguments are like variables and can be used anywhere within the program as a variable name.
Another tricky statement used in this program is:

```
while((p=in1.read()) != -1)
{
...
}
```

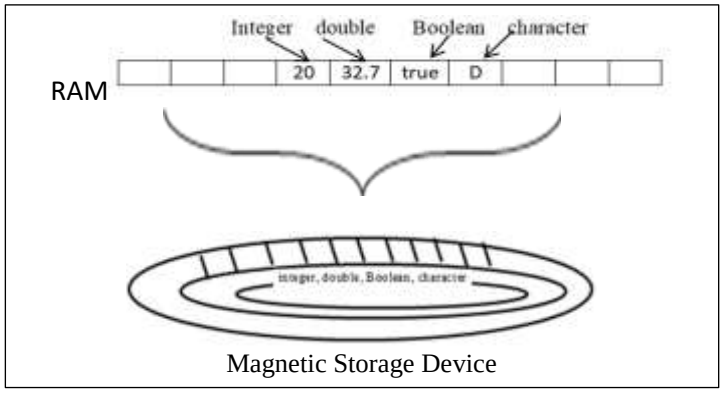
The read() method reads a character and returns its ASCII value, so, the parameter 'p' contains a number or when it goes at end of file it returns '-1'.
The logic of this statement is that if parameter value of while() is integer number then the loop will continue, but, if it is negative number as while(-1) then, the loop will exit. The concept is as:

while(0) ; loop will continue
while(1); loop will continue
……..
while(n); loop will continue
while(-1); loop will exit to out of loop

Another important statement is:
System.out.print((char) p);
Here, first the ASCII number of 'p' is converted to its corresponding character by the conversion statement '((char) p)'; so, as output we shall see the characters on the screen as shown in output of the program.

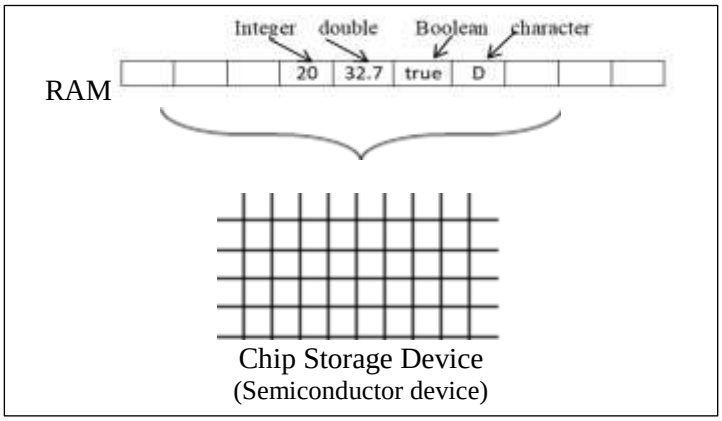
Creating data file with different data types:

In our real life, we need to store data of different properties of an object in a data file. As example, marks sheet of students where name, marks, date, pass or fail etc. different types of data are required to save in a same file. To develop that concept, here, below, a program is shown which will write different types of data in a data file and then it will read all those data and print on the screen. The main point to follow is that sequence of writing data, i.e., if we write integer, double, Boolean, character, by this order, at the time of reading data we have to follow the same sequence i.e. integer, double, Boolean, character. A graphical representation of storing data in magnetic devices(HD, FD, Tape) is shown below. First data is stored in RAM of computer and then saved within the Track and sectors of a Hard Disk or any magnetic plate.



Magnetic Storage Device

Now a days, uses of magnetic devices are decreasing and uses of Chips are increasing. Chips are used in ATM card, SIM of mobile phone, internal memory and SD card of mobile phone, RAM/ROM of Laptop etc. Very soon magnetic devices will be abandoned due to old technology and semiconductor based memory storage will be everywhere of electronic instruments. Pen drive is mostly used in computer as data storage device. Chip is an array of very small cells which can't be seen in eye. It is like Bees hub, small cells arranged in rows and columns and there are Read-Write lines connected with all cells as shown in next figure.



Now, in this example, a data file is created, then file handler is created to write and read data to/from that data file. Different types of data can be saved separately and demonstrated in this example. But, the sequence of writing data should be maintained at the time of reading data, it means, as example, if we write data of different type as: int, double, Boolean and char; then, at the time of reading data from the file, it should be int, double, Boolean and char. This sequence is important else, data will be corrupted. The technique of writing data is using a write method as writeInt() and reading an integer is readInt().

```
//FileRWall.java:  to write different types of data in a same data file and then read all those //data from the file and close the data file.

import java.io.*;
class FileRWall
{
	public static void main(String[] args) throws IOException
	{
		File in= new File("marks2.dat");
		FileOutputStream Fo= new FileOutputStream(in);
		DataOutputStream dout= new DataOutputStream(Fo);
		//Now writing data to the data file 'marks2.dat'
		dout.writeInt(2021);
		dout.writeDouble(4532.72001);
		dout.writeBoolean(true);
		dout.writeChar('D');
		dout.close();
		Fo.close();
	//reading data from the data file 'marks2.dat'
		FileInputStream fin=new FileInputStream(in);
		DataInputStream din=new DataInputStream(fin);
		System.out.println(din.readInt());
		System.out.println(din.readDouble());
```

```
                System.out.println(din.readBoolean());
                System.out.println(din.readChar());
                din.close();
                fin.close();
        }
}
```

```
E:\example>java FileRWall.java
20
32.7
true
D
```

Discussion: when we run this program, always a new file is created by name 'marks2.dat', if previous data exists, that will be deleted and a new file will be created. It is a data file, not a text file, and the data are stored by the program can't be seen by other text editor. Only program can read those data and display on the screen for the user. By careful observation of this example, you will see, for handling data another object 'dout' is created as:

```
DataOutputStream dout= new DataOutputStream(Fo);
```

In this present program, file creation class 'FileOutputStream' created file handler and data creation class 'DataOutputStream' created data handler. Printing data on the screen is done by the method 'println()' as:

```
System.out.println(din.readBoolean());
```

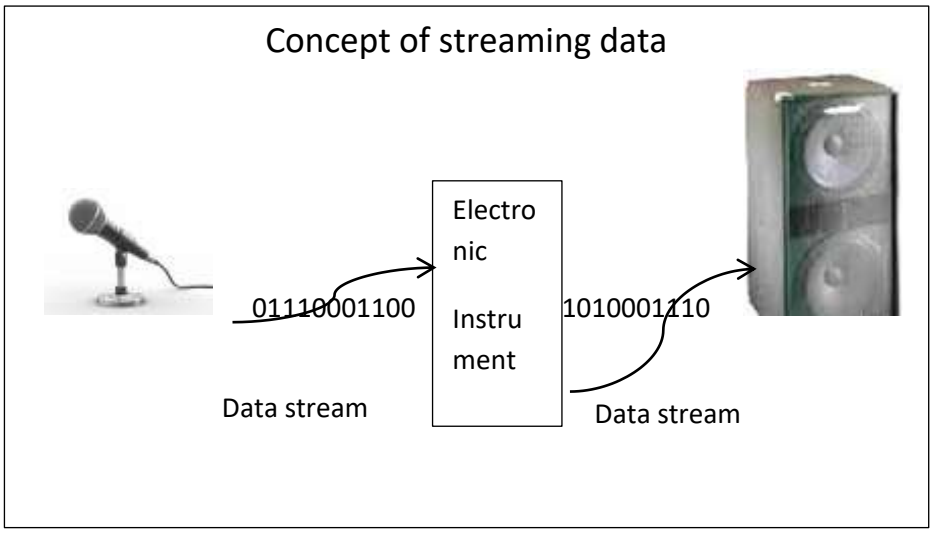
Here, first, 'readBoolean()' reads the data from the hard disk and store in RAM with the data handler 'din' meaning that it keeps track of location of this block of data in RAM and then using the method 'println()' it can display data on screen. Two jobs are performed by this way one is reading data and the other is printing data.

And the difference with creating text file is that this line was not in previous program 'FileRW.java' where we created read/write operations on text files.

Lab Work: edit the above program for changing sequence of storing different data and then read those data.

Stream of Data:

Streaming is a concept of continuous flow of data. It means without ending, the system will try to read data till end and also output data till end. But, in text read-write, data are read as a character, system controls start and end of reading of a character(i.e. 8 bits at a time), but, in stream flow, or sequence of reading-writing, data is a flow of bits not as characters. What do we mean flow of bits? In a generalized way, the concept is that when a song is sung, microphone produce electric signals, that signal is processed within electronic instruments and again sequences of signals are passed into the sound box. This flow is like a stream of wave, a continuous flow which is known as stream of data.
The concept of streaming of data is tried to explain with a figure as shown below:



But, instead of audio-video signals, if characters are streamed, technology is same, coded signals flow as stream. Here, in this program two classes like 'SequenceInputStream' and 'BufferedInputStream' are used to perform streaming concept. First, two text files are created with some texts as shown below. There are two stream handlers 'inB' for reading data from files and 'outB' for writing on screen.

C:\example>type data11.txt
Different phases of JAVA Programming: In Level-1: introduction, variables, identifiers, tokens, statements, constants, variables, different data types- integer, floating number, character data types, Boolean etc. with examples. How to display some text on Screen, print on new line etc. Then declaring variables of different data types, input data, display those data.

C:\example>type data22.txt
Different phases of JAVA Programming: In Level-2: Decision making and grouping of decisions, if ...else.., switch statement, ternary operator, ?: operator, breaking from the group, close the switch statement. Different Loop statement: while, do.., for..etc.
Concept of Class, object, instances etc.

```
//FileBuffer.java: concatenating two files through bufferstream.
// it is one type of appending data of 2nd file with 1st file.

import java.io.*;
class FileBuffer
{
        public static void main(String[] args) throws IOException
        {
                FileInputStream f1=null;
                FileInputStream f2=null;
                //a new file f3 is declared
                SequenceInputStream f3=null;
                f1=new FileInputStream("data11.txt");
                f2=new FileInputStream("data22.txt");
```

```
                f3=new SequenceInputStream(f1,f2);
                BufferedInputStream inB=new BufferedInputStream(f3);
BufferedOutputStream outB=new BufferedOutputStream(System.out);
                int ch;
                while((ch=inB.read()) != -1)
                {
                        outB.write((char)ch);
                }
                inB.close();
                outB.close();
                f1.close();
                f2.close();
        }
}
```

OUTPUT:

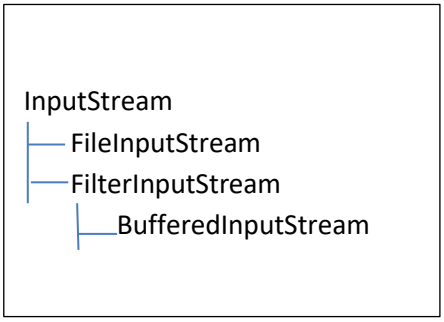
C:\example>java FileBuffer

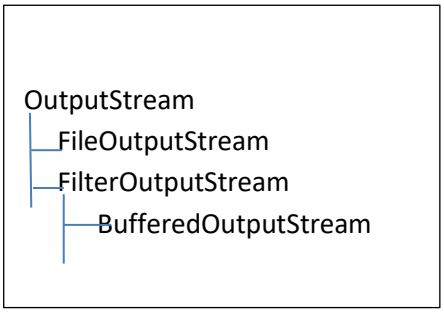
Different phases of JAVA Programming: In Level-1: introduction, variables, identifiers, tokens , statements, constants, variables, different data types- integer, floating number, character data types, Boolean etc with examples. How to display some text on Screen, print on new line etc. Then declaring variables of different data types, input data, display those data.

Different phases of JAVA Programming:In Level-2: Decision making and grouping of decisions, if ...else.., switch statement, ternary operator, ?: operator, breaking from the group, close the switch statement.

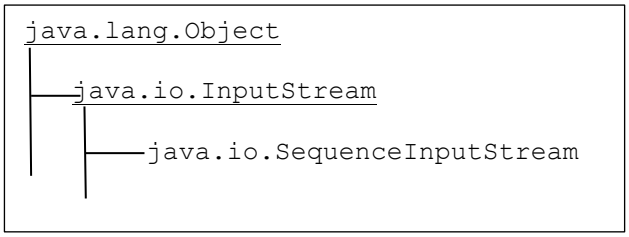
Different Loop statement:while, do.., for..etc.Concept of Class, object, instances etc.

Discussion: to test this program, first two data files should be filled with some text data. Hierarchy of **BufferedInputStream** and **BufferedOutputStream** are as:





Hierarchy of `SequenceInputStream is shown below:`



The class **SequenceInputStream** reads from any standard source till end of the data. It then closes the stream and starts to read from the second source as mentioned in program of sequence. So, we see in output, all characters are read, close the file and then characters of second file are read accordingly.

Random access of Files:

In previous examples, either read or write mode was activated and then that type of operation was done. A file can be opened as 'read mode by sysmbol r' , 'write mode by symbol w' or 'open a data file as both read-write mode'. In this example, opening a data file as read-write r-w mode is demonstrated. The basic concept of read-write data from/to a data file is that we must remember the type of data are stored and the same sequence to be read. Technology behind that is that different types of data occupies different spaces in the source of storage device. In this example, first 'Y' is saved, then an integer number '3045' is stored and so on. After writing all data to device, file is not closed. One point is required to know that when any data is written, an index pointer points to that position of the storage device. This index pointer can be moved forward backward by instructions. Here, we used 'seek(0)' to move at the beginning of the data file. Then using readChar();readInt(); etc. we have read data one by one. But, remember that we have not closed the data file after storing data, also not opened the data file for reading data. It is known as random mode and the file was opened as ("rand.dat","rw");

```
//RandomFile.java: this example shows how to read and write data in a same
//file without closing the file. There are options available for opening file as read-r,
//write-w and read-write-rw.
import java.io.*;
class RandomFile
{
        public static void main(String args[])
        {
                RandomAccessFile f1=null;
                try
                {
                        //open the data file as both read and write mode.
                        f1=new RandomAccessFile("rand.dat","rw");
                        f1.writeChar('Y');
                        f1.writeInt(3045);
                        f1.writeDouble(24.5045);
```

```
                    f1.seek(0); // point to the beginning of the file
                    System.out.println(f1.readChar());
                    System.out.println(f1.readInt());
                    System.out.println(f1.readDouble());
                    f1.seek(2); // point to the 2nd item/data
                    System.out.println((f1.readInt()));
                    f1.seek(f1.length());// goto the end of file
                    f1.writeBoolean(true);
                    f1.seek(4);// goto the 4th item lastly stored.
                    System.out.println(f1.readBoolean());
                    f1.close();
                }
                catch(IOException e) {System.out.println(e);}
        }
}
```

OUTPUT:

```
E:\example>java RandomFile
Y
3045
24.5045
3045
true
```

Discussions: many jobs of file operations have been shown in this program. These jobs are:

i. How to open a Randomaccess data file
ii. How to write/save data within randomaccess file.
iii. How to move file indicator/index pointer over the data.
iv. How to jump to a particular data position
v. How to count total length of a data file.
vi. How to junp at beginning or end of a data file
vii. How to close a data file.

These are the very important questions related to data file management. Answers are given below:

i. Generally a data file is opened as ‘r’-read mode and as ‘w’-write mode; but, if we need to want the data file as random (read-write) mode we have to open the data file as “rw” : read-write mode as shown below:

```
f1=new RandomAccessFile("rand.dat","rw");
```

ii. When a datafile is declared as “rw” mode which is known as random access file; for writing a particular type of data like character, integer or double- different methods are there to write data as shown below:

```
f1.writeChar('Y');
f1.writeInt(3045);
f1.writeDouble(24.5045);
```

Computer can mesmerize the sequence of stored data that first it was saved a character, then an integer number, then a double number. At the time of reading, pointer should point at beginning of that data and then can read the full data as example, an integer of four digits were saved, so, at the time of reading it will read four digits only not more or not less digits.

iii. How to move pointer over data: internally there is a pointer which moves over the data(but in Java pointer concept is not there), when a data is read it resides at beginning of the next data. So, when f1.readChar(),f1.readInt() or f1.readDouble() methods are used to read data it can read the all bits stored for that data type. Generally, reading a data, pointer resides at the beginning of the next data. After reading a data pointer automatically moves to the next data.

iv. Generally read(), write() methods jumps to next data. There is another method seek() can jump in front or back positions of a data. As example- this statement can jump to next 2nd data position. As written in previous program:
f1.seek(2); // from current position jumps to next 2nd data beginning
f1.seek(0);// points to beginning of the file
f1.seek(f1.length());//points to at end of file

v. In the above program, f1, is the data file handler; so f1.length() will return the size in bytes of the file which is very important to know whether the file is empty or not. If not empty, then it is required to know the size for getting idea of memory size.

vi. How to jump back and front from a given position: file pointer remains at a position over the data, if it is required to move forward then instructions are given like: f1.seek(2); which will move two data ahead; if it is integer/double – it will move so many bits ahead from the current position. The seek() method can track this situations of lengths of different data types. If seek(-2) contains minus parameter, it will jump backward accordingly.

vii. When working of read/write is done, the best practice is to close the file for safety. The statement f1.close(); will close the data file which is linked with 'f1'; the file handler. By this closing, the memory occupied by the file handler 'f1' will be released for other operations.

```
//RandomAppend.java: this file open the text file as read and write mode
//appends data at end of the file. It open previous file with its data and
//appends at end of the file.
import java.io.*;

class RandomAppend
{
	static public void main(String args[])
	{
		RandomAccessFile rf;
		try
		{
			rf=new RandomAccessFile("states.txt","rw");
			rf.seek(rf.length());
			rf.writeBytes("ASSAM\n MAHARASTRA\n");
```

```
                    rf.close();
                }
                catch(IOException e)
                {
                    System.out.println(e);
                }
        }
}
```

After two times of run of the program, the contents of the data file 'country.txt' is:

```
E:\example>type states.txt
ASSAM
 MAHARASTRA
ASSAM
 MAHARASTRA
```

```
//database.java: by this program, first a data file is opened by name 'shop.dat', then
//input three values of a medicine, as serial number, quantity and unit price, then
//these three contents are stored in the data file. At last, read three values from the
//data file, calculate total value, discount and the payable amount of a sales.
import java.io.*;
import java.util.*;

class database
{
        static DataInputStream di=new DataInputStream(System.in);
        static StringTokenizer st;

        public static void main(String args[]) throws IOException
        {
                DataOutputStream dout=new DataOutputStream(new
                FileOutputStream("shop.dat"));
                //reading from Keyboard
                System.out.println("Medicine Serial number: ");
                st=new StringTokenizer(di.readLine());
                int slno=Integer.parseInt(st.nextToken());
                //input quantity
                System.out.println("How many quantity?");
                st=new StringTokenizer(di.readLine());
                int qty=Integer.parseInt(st.nextToken());
                //input unit price
                System.out.println("Enter Unit Price:");
                st=new StringTokenizer(di.readLine());
                double price=new Double(st.nextToken()).doubleValue();
```

```
                    //data is to be stored in file 'store.dat'
                    dout.writeInt(slno);
                    dout.writeInt(qty);
                    dout.writeDouble(price);
                    dout.close();
                    //Read data from data file
                    DataInputStream dis=new DataInputStream(new
                                                FileInputStream("shop.dat"));
                    int codeno=dis.readInt();
                    int quantity=dis.readInt();
                    double unitcost=dis.readDouble();
                    double total=quantity*unitcost;
                    double discount=(total*8)/100; // 8% discount
                    dis.close();
                    //display output
                    System.out.println();
                    System.out.println( "Sl No :  " + codeno);
                    System.out.println( "Unit Price : " + unitcost);
                    System.out.println(" Total Quantity : " + quantity);
                    System.out.println(" Total Price : " + total);
                    System.out.println(" Payable amount :" + (total-discount));

          }
}
```

OUTPUT:

```
E:\example>java database
Medicine Serial number,in int?
22
How many quantity?
10
Enter Unit Price:
500

Sl No :  22
Unit Price : 500.0
 Total Quantity : 10
 Total Price : 5000.0
 Payable amount :4600.0
```

Discussions: this example is prepared to demonstrate all features of file handling as input and output jobs. First of all two *static* objects are created as ‘di’ and ‘st’ meaning that these names (di or st) can’t be reasssigned for other purposes. A new class is used by name ‘StringTokenizer’ which contain many methods for handling string operations for different tokens. But, in this example, we have used this class for inputting serial number, then we convert the string token into integer value as:

```
int slno=Integer.parseInt(st.nextToken());
```

This is nothing but to show uses of the class 'StringTokenizer'. In different ways how we can use this class that is demonstrated. The indicator of tokens 'st' will indicate the next token as second inputted data. The advantage is that by only one indicator 'st' we can identify different tokens one after another, only jumps to next tokens. Second token is converted as double data type by this line:

```
double price=new Double(st.nextToken()).doubleValue();
```

Many advanced techniques are used in this program and presented here for academic interest of learning. One point is must that if you open once a file, should close before exit the program and it is done in these ways:

```
dout.close();
dis.close();
```

Real Life Data Form:

In this example, a simple form with three columns are shown. If three columns can be created, then any numbers of columns can be extended using the same logic. Then, two push buttons 'Input' and 'Exit' are created to input data again and again, finally exit from the program. To create real life features, more programming codes are required to write. Some features are demonstrated in this program as shown below:

```
//DataEmp.java: It is a demo of a small form to input employee code (as integer),
name as String, salary as double. After input of data in any field, Tab key to be
pressed to move to next fields. For saving data in datafile, press INPUT, to Quit press
EXIT.
import java.io.*;
import java.awt.*;
class DataEmp extends Frame
{
        //Frame of a window is created
        TextField empcode, name, salary;
        Button INPUT, EXIT;
        Label codeLabel, nameLabel, salaryLabel;
        DataOutputStream douts;
        //Frame
        public DataEmp()
        {
                super("Create Payroll of Employee");
        }
        //setup the window
        public void setup()
        {
                resize(50,25);
                setLayout(new GridLayout(4,2));
```

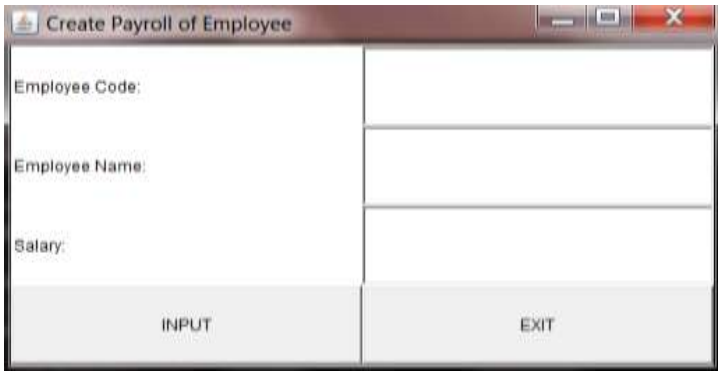
```
        //create components of the Frame
        empcode=new TextField(35);
        // Employee code is in integer
        codeLabel=new Label("Employee Code :");
        name=new TextField(35);
        nameLabel=new Label("Employee Name:");
        salary=new TextField(35);
        salaryLabel=new Label("Salary:");
        INPUT=new Button("INPUT");
        EXIT=new Button("EXIT");
        //add the component to the Frame
        add(codeLabel);
        add(empcode);
        add(nameLabel);
        add(name);
        add(salaryLabel);
        add(salary);
        add(INPUT);
        add(EXIT);
        //show the frame
        show();
        //open the file
        try
        {
                douts=new DataOutputStream(new
                        FileOutputStream("employee.dat"));
        }
        catch (IOException e)
        {
                System.err.println(e.toString());
                System.exit(1);
        }
}//end of setup
        //write to the file
        public void addRecord()
        {
                int num;
                Double d;
                num = (new Integer(empcode.getText())).intValue();
                try
                {
                        douts.writeInt(num);
                        douts.writeUTF(name.getText());
                        d=new Double(salary.getText());
                        douts.writeDouble(d.doubleValue());
```

```
            }
            catch(IOException e){}
            //clear the text field
            empcode.setText("");
            name.setText("");
            salary.setText("");
}
//adding the record and clearing the field
public void cleanup()
{
            if(!empcode.getText().equals(""))
            {
                        addRecord();
            }
            try
            {
                        douts.flush();
                        douts.close();
            }
            catch(IOException e){}
}
//processing the event
public boolean action(Event event, Object o)
{
            if(event.target instanceof Button)
            {
                        if(event.arg.equals("INPUT"))
                        {
                                    addRecord();
                                    return true;
                        }
            }
            return super.action(event, o);
}
public boolean handleEvent(Event event)
{
            if(event.target instanceof Button)
            {
                        if(event.arg.equals("EXIT"))
                        {
                                    cleanup();
                                    System.exit(0);
                                    return true;
                        }
            }
```

```
            return super.handleEvent(event);
        }
        //execute the program and the main module
        public static void main(String args[])
        {
            DataEmp employee = new DataEmp();
            employee.setup();
        }
}
```

OUTPUT:

The layout of the data input screen is shown below:



Discussions: the new things, three classes, we have added in this programs are:

TextField: to work with text within any label of a column

Button: to work with push button

IOException: to work with input/output errors

In the program, two user defined methods are created as DataEmp() and setup(). Within the program, these two lines may look like estrange, but, has internal meanings:

```
INPUT=new Button("INPUT");
EXIT=new Button("EXIT");
```

Descriptions are like these: In first line, “INPUT” within quotation is the text which will be displayed on the push button. But, at left most INPUT without quotation mark is really an user defined variable which holds the starting address of the text “INPUT” is stored within the memory of the computer. Two versions of INPUT in right and left are fully two things. In the same way, EXIT of right side and left side of the second line should be described. Another new command is:

```
douts.writeUTF(name.getText());
```

Here, the method 'writeUTF()' writes the text in Unicode Translation Format(UTF). It is a coding pattern of texts. There are three UTF coding and these are, UTF-8, UTF-16, UTF-32 of different sizes of formats.
Exception handling is one of the advanced topic and should be included within the program when input/output file handling operations are performed. In this program, it is included as:

```
catch (IOException e)
                {
                        System.err.println(e.toString());
                        System.exit(1);
                }
```

These lines of code will catch the error and print as per system's error message, finally the program will exit to the operating system, i.e., java program will be unsuccessfully terminated. In java, System.exit(0) means successful termination and System.exit(1) means unsuccessful termination of the program.

Finally, the main() method of the program contains only two lines as:

```
DataEmp employee = new DataEmp();
                        employee.setup();
```

Here, 'DataEmp' is the user defined class and at first of the program we have created it and the last '}' has closed the class body. Within main(), 'employee' is the object created from the newly created class 'DataEmp' and from that object setup() method has called to run the program. The user defined method 'setup()' contains all necessary elements to run and execute the program.

Questions and Answers:

1. What is data? Why is it required to save in storage media?
2. What are the differences between Text file and Data File?
3. Mention any two extension names of audio files.
4. Mention any two extension names of video files.
5. Mention a name of a method which can open a file.
6. What is the value in hexadecimal $(65)_{10} = (?)_{16}$
7. What are the 16 digits in hexadecimal number system?
8. Describe the instruction: `DataInputStream din=null;`
9. Describe the instruction: `rf=new RandomAccessFile("states.txt","rw");`
10. Describe the instruction: `douts.writeUTF(name.getText());`

B.1. What will be stored by this command `dout.writeBoolean(true);` ?

B.2. This command will put the indicator at what position of the file?
`rf.seek(rf.length());`

B.3. Sequence of stream is required to manage multiple files. What is the role of the command as mentioned below?

```
f3=new SequenceInputStream(f1,f2);
```

B.4. Where this command will point to?

```
f1.seek(4);
```

B.5. What will happen if we give double quote as "D" in this command as shown below?

```
dout.writeChar('D');
```

C. Multiple Choices Questions:

C.1. In text file, which format is used to store data:
a. in ASCII format
b. in UTF-8 format
c. in WAV format
d. None of the above

C.2. What will be the output of the command:

```
System.out.printf("ASCII code in Decimal is %d   of %<c \n", 97);
```

a. ASCII code in Decimal is 61 of "A"
b. ASCII code in Decimal is 97 of "a"
c. ASCII code in Decimal is "a" of 97
d. ASCII code in Decimal is 61 of "a"

C.3. Reason of closing a data file as `din.close();`

a. Data file will not be encrypted.
b. Data file may be damaged.
c. Program will not be compiled.
d. Compilation error will happen.

C.4. The best answer of this instruction is: `FileWriter out1=null;`

a. Object out1 is assigned with null value.
b. Object out1 is created from FileWriter and its address is assigned as null.
c. Object out1 is equal to zero.
d. All of the above.

C.5. This instruction, `catch(IOException e){}`, is used for

a. Printing error after compilation.
b. Printing error after execution of the program.
c. To hold input/output error within the object "e".
d. To catch all errors in the program.

Answers :

C.1. a; C.2.d; C.3. b; C.4. b; C.5. c;

Conclusion:

Within 10 chapters, we have covered most of the important features of JAVA programming; discussed theoretical issues with the help of examples of program, output of all programs are presented that readers can understand how the programs will behave. At end of each program, discussions have been included to describe new commands added in that program which will help readers to understand new features with its practical effects. This book is aimed for the beginners and middle range readers who want to learn programming of JAVA.

Link of PPT of this book and all source codes:

https://drive.google.com/drive/folders/1bMxMCaqPe0W35COAdn_-BrnV_uzQ-tNO?usp=sharing
(Note: to get access, copy and paste this link to your browser. Both PPT and PDF version of presentation of this book are uploaded in Google Drive.)

END

About the Author:

Dulal Acharjee: he is retired Professor of Computer Science and Engineering and Information Technology subjects. At present he is working as the Director of 'Applied Computer Technology', Agarpara, Kolkta, West Bengal, India. He also worked as visiting Faculty in Maulana Abul Kalam Azad University of Technology, Kolkata. He is M.Tech in Information Technology from Tezpur University, Assam, India; before, he did M.Sc. in Applied Physics and Electronics from Dhaka University. He has 30 years of teaching experiences in different College and Universities. He has published about 40 number of research papers published in different indexed journals, 4 book chapters, 10 edited Books and editor of many special issues of SCI indexed journals. He is Associate Editor and Lead Guest Editor of the Journal of Microsystem Technologies, Springer-Nature, SCI indexed. He is working as the Executive Chairman of many international conferences of science and engineering subjects.