READING DATA FROM KEYBOARD USING DATAINPUTSTREAM, BUFFEREDREADER AND SCANNER

Reading text data from keyboard using DataInputStream

As we have seen in introduction to streams, DataInputStream is a FilterInputStream that read primitive data from an underlying input stream in a machine-independent way. It implements the DataInput interface and provides convenience methods such as readInt(), readChar etc. It also inherits the methods from its immediate parent FilterInputStream.

Below program will take the input from keyboard using DataInputStream with its inherited read method and print it to a file using FileOutputStream.

Code

```
import java.io.*;


public class FileCopier {

  public static void main(String[] args) throws IOException {

    DataInputStream dis = new DataInputStream(System.in);

    FileOutputStream fout = new FileOutputStream("myFile.txt");

    System.out.println("Enter text (enter & to end):");

    char ch;

    while ((ch = (char) dis.read()) != '&')

      fout.write(ch);

    fout.close();

  }

}
```

Input will be read on every new line byte by byte and ends reading when a '&' is encountered.

FileOutputStream is a low level byte stream, whose constructor takes the name of the file as an argument. If you are not clear about streams, please do read ‘introduction to streams’ first. Here the

DataInputStream wraps the System.in variable. System class's static member variable 'in' (System.in) is of type PrintStream and represents the standard input device that is keyboard by default.

To use the methods implemented for the DataInput interface, you need to understand the contract of DataInput well or you might run into errors or confusion. Also, the readLine() method of the DataInputStream is deprecated as method does not properly convert bytes to characters. To use readLine() and other character convenience methods, a more popular approach is to use a BufferedReader with InputStreamReader.

Reading text data from keyboard using BufferedReader with InputStreamReader

Instead of a DataInputStream, you can use a BufferedReader to use methods such as readLine(). However System.in is a PrintStream object (PrintStream is an InputStream) and most readers like BufferedReader cannot directly use an InputStream. Hence you need to also use an InputStreamReader which will act like a bridge between an InputStream like PrintStream and a Reader like BufferedReader.

Code

```
import java.io.*;

public class FileCopier {

  public static void main(String[] args) throws IOException {

    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

    FileOutputStream fout = new FileOutputStream("myFile.txt");

    System.out.println("Enter text (enter string 'end' on a line to end):");

    String str = "";


    while (!str.equals("end")) {

      str = br.readLine();

      fout.write(str.getBytes());

      fout.write(System.lineSeparator().getBytes());

    }

    fout.close();

    br.close();
```

```
  }
}
```

I have only replaced DataInputStream with BufferedReader from the previous example, and kept the FileOutputStream for writing to file, as is. FileOutputStream operate with bytes, but readLine() of BufferedReader returns a String. Hence you need to convert your String to bytes using String's getBytes method. You can use a FileWriter or a BufferedWriter to write strings to the file without any casting or explicit conversion.

Code

```
import java.io.*;

public class FileCopier {

  public static void main(String[] args) throws IOException {

    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

    FileWriter fw = new FileWriter("myFile.txt");

    System.out.println("Enter text (enter string 'end' on a line to end):");

    String str = "";

    while (!str.equals("end")) {

      str = br.readLine();

      fw.write(str);

      fw.write(System.lineSeparator());

    }

    fw.close();

    br.close();

  }

}
```

Note that you no longer has to use getBytes() method of the String. Using a BufferedWriter will still improve the code in terms of performance and also adds a convenience method newLine() which will print the line separator string as defined by the system property line.separator. You can attach a FileWriter to a BufferedWriter as:

BufferedWriter bw = new BufferedWriter(fw);

And then replace the line:

fw.write(System.lineSeparator());

with

bw.newLine();

Using Scanner to read text data from keyboard

Use of Scanner is a more convenient and preferred approach for reading text data from keyboard. A Scanner can parse primitive types and strings using regular expressions. A Scanner breaks its input into tokens using a delimiter pattern, which by default is whitespace, and then get those using various next methods. You can get the whole line using the nextLine() method, which will return the rest of the current line, excluding any line separator and advances this scanner past the current line so that a call to nextLine() again will return the next line and so on.

For reading data from keyboard, we can attach the PrintStream System.in which reprecents the default input device (which is usually keyboard).

Code

```
import java.io.*;

import java.util.Scanner;


public class FileCopier {

  public static void main(String[] args) throws IOException {

    Scanner sc = new Scanner(new InputStreamReader(System.in));

    FileWriter fw = new FileWriter("myFile.txt");

    System.out.println("Enter text (enter string 'end' on a line to end):");

    String str = "";

    while (!str.equals("end")) {

      str = sc.nextLine();

      fw.write(str + "\n");
```

```
    }
    fw.close();
    sc.close();
  }
}
```

Scanner can do lot more than these and we will see them in a separate note.

Source : http://javajee.com/reading-data-from-keyboard-using-datainputstream-bufferedreader-and-scanner