Module 5

I/O Basics

Java programs perform I/O through streams. A *stream* is a logical device that either produces or consumes information. A stream is linked to a physical device by the Java I/O system. All streams behave in the same manner, even if the actual physical devices to which they are linked differ. Thus, the same I/O classes and methods can be applied to any type of device. Java defines two types of streams: byte and character. *Byte streams* are used for reading or writing binary data. *Character streams* provide a convenient means for handling input and output of characters.

Reading Console Input

In Java, console input is accomplished by reading from **System.in**. To obtain a character based stream that is attached to the console, wrap **System.in** in a **BufferedReader** object. **BufferedReader** supports a buffered input stream. Its most commonly used constructor is shown here:

BufferedReader(Reader *inputReader*)

Here, *inputReader* is the stream that is linked to the instance of **BufferedReader** that is being created. To obtain an **InputStreamReader** object that is linked to **System.in**, use the following constructor:

InputStreamReader(InputStream *inputStream*)

Because **System.in** refers to an object of type **InputStream**, it can be used for *inputStream.*

Putting it all together, the following line of code creates a **BufferedReader** that is connected to the keyboard:

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

After this statement executes, **br** is a character-based stream that is linked to the console through **System.in**. To read a character from a **BufferedReader** , we use ***read()*** method. Each time that **read( )** is called, it reads a character from the input stream and returns it as an integer value. It returns –1 when the end of the stream is encountered.

```
import java.io.*;
class BRRead
{
   public static void main(String args[]) throws IOException
   {
      char c;
     BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

       System.out.println("Enter characters, 'q' to quit.");

      do
      {
            c = (char) br.read();
            System.out.println(c);
        } while(c != 'q');
    }

}
```

Sample Output:

```
Enter characters, 'q' to quit.
abcdjqmn
a
b
c
d
j
q
```

The above program allows reading any number of characters and stores them in the buffer. Then, all the characters are read from the buffer till the ‘q’ is found and displayed.

In Java, the data read from the console are treated as strings (or sequences of characters). So, if we need to read numeric data, we need to parse the string to the respective numeric type and use it later in the program. Following is a program to read an integer value.

```
import java.io.*;
class BRRead
{

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

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

      System.out.println("Enter a number:");

      x=Integer.parseInt((br.readLine()).toString());


      x=x+5;
      System.out.println(x);
  }

}
```

Writing Console Output

Console output can be achieved using ***print()*** and ***println()*** methods. The ***PrintStream*** class provides another method ***write()*** which is capable of printing only low-order 8-bit values.

Ex:

```
int b;
b = 'A';
System.out.write(b);
System.out.write('\n');
```

PrintWriter Class

PrintWriter is one of the character-based classes. **System.out** is used to write stream of bytes. As there is a limitation for size of bytes, for most generic program (that supports various languages in the world), it is better to use PrintWriter class object to display the output. We can decide whether to flush the stream from the buffer after every newline by setting 2nd argument of the PrintWriter class constructor as ***true.***

```
import java.io.*;
public class PrintWriterDemo
{
  public static void main(String args[])
  {
    PrintWriter pw = new PrintWriter(System.out, true);
    pw.println("This is a string");

    int i = -7;
    pw.println(i);
    double d = 4.5e-7;
    pw.println(d);
  }
}
```

Reading and Writing Files

Java provides a number of classes and methods that allow you to read and write files. In Java, all files are byte-oriented, and Java provides methods to read and write bytes from and to a file. However, Java allows you to wrap a byte-oriented file stream within a character-based object. Two classes used:

FileInputStream
FileOutputStream

To open a file, you simply create an object of one of these classes, specifying the name of the file as an argument to the constructor. Two constructors are of the form:

FileInputStream(String *fileName) throws FileNotFoundException*

FileOutputStream(String *fileName) throws FileNotFoundException*

Here, *fileName specifies the name of the file that you want to open. When you create an* input stream, if the file does not exist, then **FileNotFoundException** is thrown**.** For output streams, if the file cannot be created, then **FileNotFoundException** is thrown**.** When an output file is opened, any preexisting file by the same name is destroyed. When you are done with a file, you should close it by calling **close( ).** To read from a file, you can use a version of **read( )** that is defined within FileInputStream. To write data into a file, you can use the **write( )** method defined by FileOutputStream.

Program to read data from a file:
Note to students: First create a file(using Notepad) with name "test.txt" and save it in the folder (same place where you are going to keep your Java programs). Write some contents into this file. Then create a Java program as shown below –

```
import java.io.*;

class ReadFile
{
  public static void main(String args[]) throws IOException
  {
      int i;
      FileInputStream f;

      try
      {
        f = new FileInputStream("test.txt");
      } catch(FileNotFoundException e)
      {
            System.out.println("File Not Found");
            return;
      }

      do
      {
           i = f.read();
           if(i != -1)
            System.out.print((char) i);
      } while(i != -1);

      f.close();
     }

}
```

When you run the above program, the contents of the *“test.txt”* file will be displayed. If you have not created the *test.txt* file before running the program, then the “File Not Found” exception will be caught.

Program to write data into a File:
To read the data from a file, it is obvious that the file must already exist in readable format. But, to write a data into a file, the file need not exist in the folder. Instead, when the statement to open a file to write the data is encountered in the program a file with specified name will be created. The data written will be then stored into it. If the specified file already exists inside the folder, it will be overwritten by the new data. Hence, the programmer must be careful.

Consider the below given program:

```
import java.io.*;

class WriteFile
{
   public static void main(String args[]) throws IOException
   {
     int i;
     FileOutputStream fout;
     char c;

    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
     System.out.println("Enter characters, 'q' to quit.");

     try
     {
      fout = new FileOutputStream("test1.txt");
     } catch(FileNotFoundException e)
     {
       System.out.println("Error Opening Output File");
       return;


         }


     do
     {
        c = (char) br.read();
        fout.write((int)c);
    } while(c != 'q');
  }

}
```

When you run the above program, it will ask you to enter a few characters. Give some random characters as an input and provide ‘q’ to quit. The program will read all these characters from the buffer and write them into the file *“test1.txt”*. Go to the folder where you have saved this program and check for a text file *“test1.txt”*. Open the file manually (by double-clicking on it) and see that all characters that you have entered are stored in this file.

Stream Class	Meaning
BufferedInputStream	Buffered input stream
BufferedOutputStream	Buffered output stream
ByteArrayInputStream	Input stream that reads from a byte array
ByteArrayOutputStream	Output stream that writes to a byte array
DataInputStream	An input stream that contains methods for reading the Java standard data types
DataOutputStream	An output stream that contains methods for writing the Java standard data types
FileInputStream	Input stream that reads from a file
FileOutputStream	Output stream that writes to a file
FilterInputStream	Implements **InputStream**
FilterOutputStream	Implements **OutputStream**
InputStream	Abstract class that describes stream input
ObjectInputStream	Input stream for objects
ObjectOutputStream	Output stream for objects
OutputStream	Abstract class that describes stream output
PipedInputStream	Input pipe
PipedOutputStream	Output pipe
PrintStream	Output stream that contains **print( )** and **println( )**
PushbackInputStream	Input stream that supports one-byte "unget," which returns a byte to the input stream
RandomAccessFile	Supports random access file I/O
SequenceInputStream	Input stream that is a combination of two or more input streams that will be read sequentially, one after the other

TABLE 13-1 The Byte Stream Classes

Stream Class	Meaning
BufferedReader	Buffered input character stream
BufferedWriter	Buffered output character stream
CharArrayReader	Input stream that reads from a character array
CharArrayWriter	Output stream that writes to a character array
FileReader	Input stream that reads from a file
FileWriter	Output stream that writes to a file
FilterReader	Filtered reader
FilterWriter	Filtered writer

TABLE 13-2 The Character Stream I/O Classes

Stream Class	Meaning
InputStreamReader	Input stream that translates bytes to characters
LineNumberReader	Input stream that counts lines
OutputStreamWriter	Output stream that translates characters to bytes
PipedReader	Input pipe
PipedWriter	Output pipe
PrintWriter	Output stream that contains **print( )** and **println( )**
PushbackReader	Input stream that allows characters to be returned to the input stream
Reader	Abstract class that describes character stream input
StringReader	Input stream that reads from a string
StringWriter	Output stream that writes to a string
Writer	Abstract class that describes character stream output

TABLE 13-2 The Character Stream I/O Classes *(continued)*