Experiment No. : 09

Title: Write a program to accept a file name from user and perform read, write/append operations on it.

Objectives:

1) To learn file handling in java.

Theory:

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 of the most often-used stream classes are **FileInputStream** and **FileOutputStream**, which create byte streams linked to files. 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. While both classes support additional, overridden constructors, the following are the forms that we will be using:

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( )**. It is defined by both **FileInputStream** and **FileOutputStream**, as shown here:

void close( ) throws IOException

To read from a file, you can use a version of **read( )** that is defined within F**ileInputStream**. The one that we will use is shown here:

int read( ) throws IOException

Each time that it is called, it reads a single byte from the file and returns the byte as an integer value. **read( )** returns –1 when the end of the file is encountered. It can throw an **IOException**.

To write to a file, you will use the **write( )** method defined by **FileOutputStream**.Its simplest form is shown here:

void write(int *byteval*) throws IOException

This method writes the byte specified by *byteval* to the file. Although *byteval* is declared as an integer, only the low-order eight bits are written to the file. If an error occurs during writing, an **IOException** is thrown. The programmer must include extra program statements to determine which event actually occurred. In Java, errors are passed to your program via exceptions, not by values returned by **read( )**. Thus, when **read()** returns –1, it means only one thing: **the end of the file** has been encountered.

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, you wrap System.in in a BufferedReader object, to create a character stream. BuffereredReader 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. Reader is an abstract class. One of its concrete subclasses is InputStreamReader, which converts bytes to characters. 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 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.

Reading Characters:

To read a character from a BufferedReader, use read( ).

int read( ) throws IOException

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.

Reading Strings:

To read a string from the keyboard, use the version of readLine( ) that is a member of the BufferedReader class. Its general form is shown here:

String readLine( ) throws IOException

Key concepts: BufferedReader, FileInputStream, FileOutputStrea,FileNotFoundException.

Algorithm:

1. Create a class with name ReadFile.
2. Get name of the file from user.
3. Create FileInputStream object. Read and display contents of file using read() method.
4. Ask user if he wants to add contents to the file.
5. If yes then read the contents from the user.
6. Create object of FileOutputStream and write the data into file.
7. If file does not exist then create a new file and store user data into it.

Note: Please follow the naming conventions while writing the program.