**DESCRIPTION OF CODES ON STREAMS (Expt-2)**

**import java.util.\***

Package *java.util* contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array). \* is used to include all.

**import java.io.\***

Package *java.io* provides for system input and output through data streams, serialization and the file system. Unless otherwise noted, passing a null argument to a constructor or method in any class or interface in this package will cause a *NullPointerException* to be thrown.

**class file**

Creates a class file.

**public static void main (String s[])throws IOException**

This specifies that the method may going to throws IOException and insist the compiler that the block which is calling this method needs some special attention towards this regarding handling or again throwing back.

*throws IOException:* the logic present within the main method performs some I/O related tasks, and to make sure that the program does not crash owing to some I/O related issues, this work as a fail- safe. Alternately, wrapping the responsible code within a try- catch construct also works.

**FileOutputStream f= new FileOutputStream("123.txt")**

creates a new text file with name-123.txt. If this file is already there, it is open and ready for writing.

*FileOutputStream* is the keyword. *f* is the variable.

**Scanner sc=new Scanner(System.in)**

Here, we have created an object of Scanner named *sc*. The *System.in* parameter is used to take input from the standard input. It works just like taking inputs from the keyboard.

**ch=(char)sc.next().charAt(0)**

To read a character in Java, we use *next()* method followed by *charAt(0).* The *next()* method returns the next token/ word in the input as a string and *chatAt()* method returns the first character in that string.

**f.write(ch)**

This method writes data into the file but one symbol at a time.

**f.close()**

It closes the writing operation and save the file.

**System.out.println**

It prints a message to the standard output and appends a newline character.

**public class withBuf**

class with name *withBuf* is created.

**String filePath = "C:\\Users\\dipna\\eclipse-workspace\\test\\src\\test\\file.txt";**

Full path to the file which required to be opened. This path will vary system to system. It needs double backslashes (\\) to create a single backslash in the String, because backslash is an escape character in Java Strings. To get a single backslash you need to use the escape sequence \\.

**long startTime = System.currentTimeMillis();**

**long endTime = System.currentTimeMillis();**

**long duration = endTime - startTime;**

*long* type is used to avoid overflow. *startTime, endtime and duration* are the variables.

*System.currentTimeMillis()*method returns the difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC(coordinated universal time). Duration is evaluated by taking the interval of ending and start time

**InputStream inputStream = new FileInputStream(filePath)**

*inputStream* is a user defined variable. *InputStream* is default keyword.

File is opened using *FileInputStream*. *FileInputStream* constructor takes a File object as parameter. The File object has to point to the file you want to read. An input stream is created using *FileInputStream*. It is because *InputStream* is an abstract class. Hence, we cannot create an object of *InputStream*.

**int bytesRead;**

Variable declaration

**byte[] buffer = new byte[1024];**

*bytebuffer* is used to hold the sequence of integer values that can be used for input/output operations. It creates a buffer of 1024 bytes to store the data it reads.

**while ((bytesRead = inputStream.read(buffer)) != -1)**

The *InputStream.read()* method reads the next byte of the data from the input stream and returns int in the range of 0 to 255. If no byte is available because the end of the stream has been reached, the returned value is -1.

**InputStream inputStream = new BufferedInputStream(new FileInputStream(filePath))**

The *BufferedInputStream* reads a chunk of bytes into a byte array from the underlying *FileInputStream*. You can then read the bytes one by one from the *BufferedInputStream* and still get a lot of the speedup that comes from reading an array of bytes rather than one byte at a time.

**catch (IOException e)**

*IO* is the type of exception, *e* is the variable.

**printStackTrace()**

It is very useful in diagnosing exceptions. For example, if one out of five methods in your code cause an exception, *printStackTrace()* will pinpoint the exact line in which the method raised the exception. The *printStackTrace()* method in Java is a tool used to handle exceptions and errors. It is a method of Java’s throwable class which prints the throwable along with other details like the line number and class name where the exception occurred.