CHAPTER 1

[JAVA INPUT & OUTPUT API]

[SEN2321]

PRINCIPLES OF PROGRAMMING II

SULEIMAN HAYAT TUGE

CHAPTER DURATION 2 WEEKS

# WEEK 1

## JAVA INPUT & OUTPUT

Java IO is an API that comes with Java which is targeted at reading and writing data (input and output). Most applications need to process some input and produce some output based on that input. For instance, read data from a file or over network, and write to a file or write a response back over the network.

The Java IO API is located in the Java IO package (java.io). If you look at the Java IO classes in the java.io package the vast number of choices can be rather confusing. What is the purpose of all these classes? Which one should you choose for a given task? How do you create your own classes to plugin? etc.

The purpose of this lecture is to try to give you an overview of how all these classes are grouped, and the purpose behind them, so you don't have to wonder whether you chose the right class, or whether a class already exists for your purpose.

**Streams**

A stream is a conceptually endless flow of data. You can either read from a stream or write to a stream. A stream is connected to a data source or a data destination. Streams in Java IO can be either byte based (reading and writing bytes) or character based (reading and writing characters).

Java's IO package mostly concerns itself with the reading of raw data from a source and writing of raw data to a destination. The most typical sources and destinations of data are these:

1. **Files**
2. Pipes
3. **Network Connections**
4. In-memory Buffers (example arrays)
5. **System.in, System.out, System.error**

In the last two semester we have discussed on the 5th IO item that is System.in, System.out. Below is a refresher example of what we have done. In this class we will discuss on Files and Network Connections.

## DISPLAYING MESSAGES

To display messages to the user on console the programmer uses either println or print function, The println terminates the current line by writing the line separator string. while the print method just prints the given content.

**Sample Code:**

package displayproject;

/\*\*  
 \* **@author** HSMLTUGE  
 \*/  
public class **DisplayProject** {  
 public static void ***main***(String[] args) {

System.*out*.println("Hello Java");

System.*out*.print("Why are you this simple?");

System.*out*.println("Can we have fun together?");

}  
 }

**Code Output:**

> Hello Java

> Why are you this simple?Can we have fun together?

>

## RECEIVING INPUT FROM USER

The Scanner class is used to get user input, and it is found in the java.util package. To use the Scanner class, the programmer creates an object of the class and use any of the available methods found in the Scanner class documentation [nextBoolean, nextByte, nextDouble, nextInt, nextFloat, nextLine, nextShort or nextLong]. In our example, we will use the nextInt() method, which is used to read integer value, :

**Sample Code:**

package displayproject;

import java.util.Scanner;  
/\*\*  
 \* **@author** HSMLTUGE  
 \*/  
public class **DisplayProject** {  
 public static void ***main***(String[] args) {

Scanner scan = new Scanner(System.*in*);

int age = scan.nextInt();

}  
 }

**Code Output:**

The code above shows a cursor pointer flickering, waiting for input from the user, If the user should type a number and press the enter key, the application will show nothing in the console but the value is stored in the computer memory.

From the above two examples students will realize that it takes the combination of both inputs and outputs to makes computer programs (software) interactive.

## SHOWING USER’S INPUT

For your application to be interactive, you need to pass instructions to the user, these instructions allow the user to know the state of your software. Looking at sample code 2, the user will not know what to do since he/she is not asked to do anything. But with instructions in place the user will know exactly what to do and when to do it.

**Sample Code:**

package displayproject;

import java.util.Scanner;  
/\*\*  
 \* **@author** HSMLTUGE  
 \*/  
public class **DisplayProject** {  
 public static void ***main***(String[] args) {

System.*out*.println("How old are you?");

Scanner scan = new Scanner(System.*in*);

int age = scan.nextInt();  
System.*out*.println("You are " + age + " years old")

}  
 }

**Code Output:**

> How old are you?

> 10

> You are 10 years old

## RETEST OUR OLD ACQUIRED SKILLS

Now that we have revised what we discussed last two semesters, lets try writing a java program that receive an input of type string from a user and display the string one letter at a time from the last to the first. **[10 Minutes – 2marks]**

## UNDERSTANDING DIRECTORY STRUCTURE

It is a well-known fact that computers have both volatile and non-volatile memory, volatile memory is the computer memory that stores data only when the computer is powered example is computer (Random Access Memory popularly known as RAM), while non-volatile memory allows data to be stored even when the computer is not powered example is the hard drive, in some cases the hard drive of computer A can be moved to computer B and the computer will still run smoothly. For students to understand file input stream and file output streams it is paramount for the student to have a basic understanding of how files and directories are structured.

## ROOT (WINDOWS)

In DOS (Disk Operating System), the root directory is "drive letter: \", for example, the root directory is usually "C:\". The directory separator is usually a "\", but the operating system also internally recognizes a "/". Physical and virtual drives are named by a drive letter, as opposed to being combined as one.

## FOLDER

In computers, a folder is the virtual location for applications, documents, data or other sub-folders. Folders help in storing and organizing files and data in the computer. The term is most commonly used with graphical user interface operating systems, though folder can be used in DOS OS too.

## FILE

A file is a container in a computer system for storing information. Files used in computers are similar in features to that of paper documents used in library and office files. There are different types of files such as text files, data files, directory files, binary and graphic files, and these different types of files store different types of information. In a computer operating system, files can be stored on optical drives, hard drives or other types of storage devices.

# WEEK 2

## FILE INPUT & OUTPUT STREAMS

OUTPUT STREAM

Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket. Java FileOutputStream is an output stream used for writing data to a file. If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.

INPUT STREAM

Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket. Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use FileReader class.

## CREATING FILE USING JAVA

The File class is an abstract representation of file and directory pathname. A pathname can be either absolute or relative.

A path is absolute if it starts with the root element of the file system. In windows, the root element is a drive e.g. C:\, D:\, while in unix (Linux, Mac) it is denoted by “/” character. An absolute path is complete in that no other information is required to locate the file, it usually holds the complete directory list starting from the root node of the file system till reaching the file or directory it denotes.

A relative path is a path which doesn’t start with the root element of the file system. It is simply the path needed in order to locate the file from within the current directory of your program. It is not complete and needs to be combined with the current directory path in order to reach the requested file. In order to construct a rigid and platform independent program, it is a common convention to use a relative path when locating a file inside your program.

The File class have several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory etc.

**Sample Code:**

**import** java.io.\*;

**public** **class** FileDemo {

**public** **static** **void** main(String[] args) {

**try** {

             File file = **new** File("javaFile123.txt");

**if** (file.createNewFile()) {

                 System.out.println("New File is created!");

             } **else** {

                 System.out.println("File already exists.");

             }

         } **catch** (IOException e) {

             e.printStackTrace();

         }

     }

}

**Code Output:**

When this program is executed the output will be a file with the name javaFile123.txt and a display on the console New File is created!, in the event the file already exist the system the system will not overwrite the file but just display the message File already exists. In the above code student will realize the full path to the file was not provided rather the relative path is used to create the file.

## WRITING CONTENT TO FILE

Over the years software tends to write contents to file, some of the data written in file includes configuration/settings files, helper files, or log file. For our next example we will be using error log file, error log file is a file that stores all the errors encountered during the running of the software. This allows system developers to track errors generated.

**Sample Code:**

package displayproject;

import java.io.\*;

import java.text.\*;

import java.util.\*;

/\*\*

\*

\* **@author** HSMLTUGE

\*/

public class **DisplayProject** {

public static void ***main***(String[] args) {

while (true) {

try {

System.*out*.println("How old are you?");

Scanner scan = new Scanner(System.*in*);

int age = scan.nextInt();

System.*out*.println("You are " + age + " years old");

} catch (Exception e) {

try {

TimeZone tz = TimeZone.*getTimeZone*("GMT");

DateFormat df = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss'Z'");

df.setTimeZone(tz);

String nowAsISO = df.format(new Date());

FileWriter myWriter = new FileWriter("log.txt");

System.*out*.println(e);

myWriter.write(nowAsISO + " " + e);

myWriter.close();

} catch (Exception ex) {

}

}

}

}

}

**Code Output:**

Upon execution the program asks the age of the user, The program expects a numeral age, but if the user decides to enter the word instead of number, then the program will create a log file and store the time the error was encountered and the type of error encountered. Since the code is entrapped in a while(true) loop which is always true, the system will then ask for the age of the user.

If in the second attempt the user still enters a word input instead of a number, the application will not create another logfile since one already exists, and it will not append any content to the file.

Another method of writing content to file is via the file output stream.

**Sample Code:**

import java.io.\*;

/\*\*

\*

\* **@author** HSMLTUGE

\*/

public class **DisplayProject** {

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

String value = "Hello";

String filename = "log.txt";

FileOutputStream fos = new FileOutputStream(filename);

DataOutputStream outStream = new DataOutputStream(new BufferedOutputStream(fos));

outStream.writeUTF(value);

outStream.close();

}

}

**Code Output:**

This will override the existing file if the file exists and add the content hello to the file.

## READING CONTENT FROM FILE

Files are meant to be read, the content of a file give the reader a new perspective or understanding, in Java reading a file fall under the input output class. There are many ways to read a file content.

**FILE INPUT STREAM**

The file input stream allows you to read a java file, there are many methods to read a file using the file input streams, a typical example is:

**Sample Code:**

package displayproject;

import java.io.\*;

/\*\*

\*

\* **@author** HSMLTUGE

\*/

public class **DisplayProject** {

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

String filename = "log.txt";

// verify the results

String result;

FileInputStream fis = new FileInputStream(filename);

DataInputStream reader = new DataInputStream(fis);

result = reader.readUTF();

System.*out*.println(result);

reader.close();

}

}

**Code Output:**

>Hello

**FILE READER**

package displayproject;

import java.io.\*;

/\*\*

\*

\* **@author** HSMLTUGE

\*/

public class **DisplayProject** {

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

File file = new File("log.txt");

BufferedReader br = new BufferedReader(new FileReader(file));

// Declaring a string variable

String st;

// Consition holds true till

// there is character in a string

while ((st = br.readLine()) != null) // Print the string

{

System.*out*.println(st);

}

}

}

**Code Output:**

>Hello

**SCANNER**

package displayproject;

import java.io.\*;

import java.util.Scanner;

/\*\*

\*

\* **@author** HSMLTUGE

\*/

public class **DisplayProject** {

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

File file = new File("log.txt");

Scanner sc = new Scanner(file);

while (sc.hasNextLine()) {

System.*out*.println(sc.nextLine());

}

}

}

**Code Output:**

>Hello

## MODIFYING CONTENT IN FILE

## CHECK IF FILE EXISTS

It is often very important to check if a file already exists before performing any operation, this is because should the file exists then there is no need to attempt recreating the file, or overriding the file. To override a file means to delete it and add a new file, that often is problematic. Looking at our example of log file, Imaging the log file that is supposed to tell you when errors are encountered get deleted and new entry added, that means the error history will not be properly captured. But in configuration file, it becomes paramount to override the file since you will want your system to run on the latest configurations.

**Sample Code 5:**

package displayproject;

import java.io.\*;

/\*\*

\*

\* **@author** HSMLTUGE

\*/

public class **DisplayProject** {

public static void ***main***(String[] args) {

File myObj = new File("log.txt");

if (myObj.exists()) {

System.*out*.println("File name: " + myObj.getName());

System.*out*.println("Absolute path: " + myObj.getAbsolutePath());

System.*out*.println("Writeable: " + myObj.canWrite());

System.*out*.println("Readable " + myObj.canRead());

System.*out*.println("File size in bytes " + myObj.length());

} else {

System.*out*.println("The file does not exist.");

}

}

}

**Output:**

> File name: log.txt

> Path: C:\Users\HSMLTUGE\Documents\NetBeansProjects\DisplayProject\log.txt

> Writeable: true

> Readable true

> File size in bytes 25

## DELETING FILE

## CHAPTER 1 HOME WORK

Write a java GUI program that receives the name of a file, and when the save button is clicked your program should create that file with the name received.