CS202: IT Workshop

Java

File handling

Ref:

1. Harvey Deitel, Paul Deitel, **Java: How to Program**, 9/e, Prentice Hall India.
2. Internet

File and IO Stream

- **Why file?**

File and IO Stream

- **Why file?** To make data persistent
- File are stored on secondary devices and we perform several operations (i/o) on them

File and IO Stream

- We use **stream** (a sequence of elements) when dealing with input/output (support available in **java.io**)
- **Byte-based stream** v/s **character-based stream**

Byte-based	Character-based
input and output data in binary format	input and output data as a sequence of characters
E.g. 5 is stored as its numeric value (101)	E.g. 5 is stored as its Unicode representation 53 (00000000 00110101)
Referred as binary files / raw data	Referred as text files

File handling in Java

- Java program opens a file by creating an **object** and associating a **stream** with it.

```
File file = new File( filename );
```

filename is a string that specifies the name of the file along with its path

e.g.

"/home/student/hi.txt"

"D:\\student\hi.txt"

File handling in Java

- Java program opens a file by creating an **object** and associating a **stream** with it.

```
File file = new File( filename );
```

filename is a string that specifies the name of the file along with its path
e.g.
"/home/student/hi.txt"
"D:\\student\hi.txt"

- File class supports a number of methods.

```
file.getAbsolutePath();

file.exists();
```

Returns the complete pathname along with filename

Returns true if the file specified using filename exists

Code demonstration: FileHandlingDemo.java

File instance in java

- We may pass a path/directory to File constructor

```
File file = new File (“D:\\“ );
```

- File object can be used to print the contents of a directory

```
File file = null;.

String[] paths;

        file = new File("D:\\");
        paths = file.list();

        for(String path: paths) {
                System.out.println(path);
        }
```

Returns an array of strings

Code demonstration: DirectoryDemo.java

File Reading and Writing: Byte stream

- We can perform read or write to a file using streams: **FileInputStream** and **FileOutputStream**

```
FileInputStream  in = new FileInputStream ("input.txt");

FileOutputStream  out = new FileOutputStream ("output.txt");
```

File Reading and Writing: Byte stream

- We can perform read or write to a file using streams: **FileInputStream** and **FileOutputStream**

```
FileInputStream  in = new FileInputStream ("input.txt");

FileOutputStream  out = new FileOutputStream ("output.txt");
```

- FileInputStream class supports a number of methods

```
while ( ( b = in.read() ) != -1 )  {

        out.write(b);
    }
```

read() method returns one byte at a time.
It returns -1 when end of file is reached

write() puts the byte to the file using output stream

Code demonstration: FileCopy.java

Code demonstration: FileCopy.java

HW: Modify the above program so that the following from the command prompt can do the file copy operation

$ *Java FileCopy input.text output.txt*

File Reading and Writing: Character stream

- We may use FileReader and FileWriter for reading and writing one character at a time (2 bytes)

```
FileReader fr = new FileReader("D:\\testout.txt");

FileWriter fw = new FileWriter("D:\\testin.txt");
```

File Reading and Writing: Character stream

- We may use FileReader and FileWriter for reading and writing one character at a time (2 bytes)

```
FileReader fr = new FileReader("D:\\testout.txt");

FileWriter fw = new FileWriter("D:\\testin.txt");
```

- They support a number of methods

```
int ch;
while( ( ch = fr.read() ) != -1 )
    System.out.print( (char) ch );
```

read() method returns the Unicode value of one character at a time

```
fw.write("I love IIITG");
```

write() method writes a character or a string

Code demonstration: FileCharacterDemo.java

Advanced character streams

- Character streams perform operations character by character. Suppose, we want to read a complete line.
- We may take help of java.io.BufferedReader

```
File file = new File(filename);

FileReader fileReader = new FileReader(file);

BufferedReader br = new BufferedReader(fileReader);
```

- We can read a complete line and then work on that

```
String line = br.readLine();
```

Code demonstration: BufferedReaderDemo.java

Reading data from console

- BufferedReader can be used to read input from the console and then to work on that

```
InputStreamReader r=new InputStreamReader(System.in);

BufferedReader br=new BufferedReader(r);
. . .

String input = br.readLine();
```

Code demonstration: BufferedReaderSystem.java

Reading data from console

- We may also use BufferedWriter for efficient writing

```
FileWriter fileWriter = new FileWriter(file);

BufferedWriter bwr = new BufferedWriter(fileReader);
bwr.write("hello");
bwr.newLine();
```

- Buffering technique improves IO performance

Object Serialization

- A mechanism of converting the state of an object (type of the object, type of the data stored in an object) into a byte stream.
- Then the byte stream can be written to a file
- Recreating the object in memory from a byte stream is known as **Deserialization**
- **Serialization and Deserialization** are platform independent
- **ObjectInputStream** and **ObjectOutputStream** are used with *FileInputStream* and *FileOutputStream* for serializing and deserializing an object.

```
FileOutputStream fileOut = new FileOutputStream ("/tmp/employee.ser");

ObjectOutputStream out = new ObjectOutputStream (fileOut);
```

Code demonstration: SerializationDemo.java