FILE HANDLING

INTRODUCTION TO FILE:

File

A File object is used to obtain or manipulate the information associated with a disk file, such as the permissions, time, date, and directory path, and to navigate subdirectory hierarchies. A directory in Java is treated simply as a File with one additional property—a list of filenames that can be examined by the list() method.To use a file class import java.io.File.

The following constructors can be used to create File objects:

- File(String directoryPath)
- File(String directoryPath, String filename)
- File(File dirObj, String filename)
- File f1 = new File("/");
- File f2 = new File("/" , "autoexec.bat");
- File f3 = new File(f1 , "autoexec.bat");

File defines many methods that obtain the standard properties of a File object. For example, getName() returns the name of the file, getParent() returns the name of the parent directory, and exists() returns true if the file exists, false if it does not.

```
class FileDemo {
static void p(String s) {
System.out.println(s);
}
public static void main(String[] args) {
File f1 = new
File("/java/COPYRIGHT");
p("File Name: " + f1.getName());
p("Path: " + f1.getPath());
p("Abs Path: " + f1.getAbsolutePath());
p("Parent: " + f1.getParent());
p(f1.exists() ? "exists" : "does not exist");
p(f1.canWrite() ? "is writeable": "is not writeable");
p(f1.canRead() ? "is readable": "is not readable");
p("is" + (f1.isDirectory() ? "" : "not" + a directory")):
p(f1.isFile() ? "is normal file": "might be a named
pipe");
p(f1.isAbsolute() ? "is absolute": "is not absolute");
p("File last modified: " + f1.lastModified());
p("File size: " + f1.length() + " Bytes");
}
}
```

Output:

```
File Name: COPYRIGHT
Path: /java/COPYRIGHT
Abs Path: /java/COPYRIGHT
Parent: /java
exists
is writeable
is readable
is not a directory
is normal file
is absolute
File last modified: 8124
```

Directories

A directory is a File that contains a list of other files and directories. When we create a File object and it is a directory, the is Directory() method will return true. In this case, we can call list() on that object to extract the list of other files and directories inside.

```
class DirList {
public static void main(String[] args) {
String dirname = "/java";
File f1 = new File(dirname);
if (f1.isDirectory()) {
System.out.println("Directory of " +dirname);
String s[]=f1.list();
for (int i=0; i<s.length; i++) {
File f = new File(dirname + "/" +s[i]);
if (f.isDirectory()) {
System.out.println(s[i] + " is a directory");
} else {
System.out.println(s[i] + " is a file");
}
}
}
else {
System.out.println(dirname + " is
not a directory");
}
}
}
```

To limit the number of files returned by the list() method to include only those files that match a certain filename pattern, or filter.
To do this, use a second form of list():

- String[] list(FilenameFilter FFObj)

FilenameFilter(an interface) defines only a single method, accept(), which is called once for each file in a list. Its general form is:

- boolean accept(File directory, String filename)

The accept() method returns true for files in the directory specified by directory that should be included in the list (that is, those that match the filename argument), and returns false for those files that should be excluded.

```
import java.io.*;
public class OnlyExt implements FilenameFilter {
String ext;
public OnlyExt(String ext) {
this.ext = "." + ext;
}
public boolean accept(File dir, String name) {
return name.endsWith(ext);
}
}
```

```
// Directory of .HTML files.
import java.io.*;
class DirListOnly {
public static void main(String[] args) {
String dirname = "/java";
File fl = new File(dirname);
FilenameFilter only = new OnlyExt("html");
String s[] = f1.list(only);
for (int i=0; i < s.length; i++) {
System.out.println(s[i]);
}
}
}
```

- File[] listFiles()
- File[] listFiles(FilenameFilter FFObj)
- File[] listFiles(FileFilter Fobj)

  - FileFilter defines only a single method, accept(), which is called once for each file in a list. Its general form is:
    boolean accept(File path)

Creating Directories
Another two useful File utility methods are mkdir( ) and mkdirs(). The mkdir() method creates a directory, returning true on success and false on failure. Failure indicates that the path specified in the File object already exists, or that the directory cannot be created because the entire path does not exist yet. To create a directory for which no path exists, use the mkdirs() method. It creates both a directory and all the parents of the directory.

The Stream Classes
Java's stream - based I/O is built upon four abstract classes:

- InputStream, OutputStream, Reader and Writer.
- InputStream and OutputStream are designed for byte streams. Reader and Writer are designed for character streams

InputStream
InputStream is an abstract class that defines Java's model of streaming byte input. All of the methods in this class will throw an IOException on error conditions.

Method	Description
int available()	Returns the number of bytes of input currently available for reading.
void close()	Closes the input source. Further read attempts will generate an **IOException**.
void mark(int numBytes)	Places a mark at the current point in the input stream that will remain valid until numBytes bytes are read.
boolean markSupported()	Returns true if mark() / reset() are supported by the invoking stream.
int read()	Returns an integer representation of the next available byte of input. -1 is returned when the end of the file is encountered.
int read(byte buffer[])	Attempts to read up to buffer.length bytes into buffer and returns the actual number of bytes that were successfully read. -1 is returned when the end of the file is encountered.
int read(byte buffer[], int offset, int numBytes)	Attempts to read up to numBytes bytes into buffer starting at buffer[offset], returning the number of bytes successfully read. -1 is returned when the end of the file is encountered.

OutputStream
OutputStream is an abstract class that defines streaming byte output. All of the methods in this class return a void value and throw an IOException in the case of errors.

Method	Description
void close()	Closes the output stream. Further write attempts will generate an **IOException**.
void flush()	Finalizes the output state so that any buffers are cleared. That is, it flushes the output buffers.
void write(int b)	Writes a single byte to an output stream. Note that the parameter is an int, which allows us to call write() with expressions without having to cast them back to byte.
void write(byte buffer[])	Writes a complete array of bytes to an output stream.
void write(byte buffer[], int offset, int numBytes)	Writes a subrange of numBytes bytes from the array buffer, beginning at buffer[offset].

The FileInputStream class creates an InputStream that we can use to read bytes from a file. Its two most common constructors are shown here:

- FileInputStream(String filepath)
- FileInputStream(File fileObj).

Example:

- FileInputStream f0= new FileInputStream("/autoexec.bat")
- File f= new File("/autoexec.bat");
- FileInputStream f1 = new FileInputStream(f);

FileOutputStream creates an OutputStream that we can use to write bytes to a file. Its most commonly used constructors are shown here:

- FileOutputStream(String filePath)

- FileOutputStream(File fileObj)
- FileOutputStream(String filePath, boolean append)
- FileOutputStream(File fileObj, boolean append)

```
import java.io.*;
public class CopyFile {
   public static void main(String[] args) throws IOException {
      FileInputStream in = null;
      FileOutputStream out = null;
      try {
        in = new FileInputStream("input.txt");
        out = new FileOutputStream("output.txt");
        int c;
        while ((c = in.read()) != -1) {
        out.write(c);
   }
}
catch(Exception ex)
{
}
finally {
   if(in!=null)
   in.close();
   if(out!=null)
   out.close();
   }
  }
}
```

Writing and reading object demo:

-The ObjectOutputStream and ObjectInputStream provides the method to write an object to a file and read an object from a file respectively.
The constructor for ObjectOutputStream is:
public ObjectOutputStream(OutputStream out) throws IOException {
...
}
and the constructor of ObjectInputStream is:
public ObjectInputStream(InputStream in) throws IOException {
...
}

To write an object java need to store the state of an object.

Serialization: It is a process by which java stores the state of an object into the bytestream.

Deserialization: It is a process by which java restores the state of an object from the bytestream.

To write the object into the File the object class should implement Serializable or externalizable interface.

Below is the program for writing and reading object from the file by implementing the Serializable interface.

```
package FileHandle;
import java.io.*;
class Student implements Serializable {
String name,address;
int rollno;
public Student(String n, String a,int r) {
this.name=n;
this.address=a;
this.rollno=r;

}
public void printInfo() {
System.out.println("Student Info:");
System.out.println("Name:"+this.name);
System.out.println("Address:"+this.address);
System.out.println("rollno:"+this.rollno);
}
}
public class WritingObjectDemo {
public static void readObject(String filepath) {
try{
File f=new File(filepath);
FileInputStream fis=new FileInputStream(f);
ObjectInputStream ois=new ObjectInputStream(fis);
Student[] s=(Student[])ois.readObject();
for(Student ob:s){
ob.printInfo();
}
}catch (Exception e) {
System.out.println(e);
}
}
public static void writeObject(String filepath) {
try{
Student s=new Student("Ashesh", "Baneshwor",1);
Student s1=new Student("Neupane", "Baneshwor",2);
Student sarr[]={s,s1};
File f=new File(filepath);
FileOutputStream fos=new FileOutputStream(f);
ObjectOutputStream oos=new ObjectOutputStream(fos);
oos.writeObject(sarr);
}catch (Exception e) {
System.out.println(e);
}
}
public static void main(String[] args) {
// writeObject("ashesh.txt");
readObject("ashesh.txt");
}
}
```

PrintStream class:

```
import java.io.PrintStream;
import java.io.PrintWriter;
public class PrintStreamDemo {
public static void main(String[] args) {
System.out.println("hello world");
System.out.println("from nepal");
PrintStream s=System.out;
try {
PrintStream ps = new PrintStream("ashesh.txt");
ps.println("hello world");
ps.println("from Nepal");
ps.print("good");
ps.print("day");
ps.printf("The no is %d",123);
ps.close();
}catch (Exception e) {
System.out.println(e);
}
}
}
```

RandomAccessFile class:

RandomAccessFile class in java is used to access the file content from the random position.
The constructor for RandomAccessFile class is:

```
public RandomAccessFile(String name, String mode) throws FileNotFoundException
{
   ...
}
```

Here the mode should be either "r" or "w" where r means read and w means write mode.
The example is shown below:

```
package FileHandle;
import java.io.RandomAccessFile;
public class RandomAccessFileDemo {
public static void main(String[] args) {
try {
RandomAccessFile rf = new RandomAccessFile("ashesh.txt","r");
int c;
rf.seek(2);
while((c=rf.read())!=-1)
{
System.out.println((char)c);
}
}
catch (Exception e)
{
System.out.println(e);
}
}
}
```