**Files**

**--------------**

**File(I/O)**

**-----------**

java io was popular at the begining

Now database concepts

Now these are not very important, but good to know,

Not so conveninet way to store data.

Database concepts are useful

File IO concepts are useful to store small amounts of data.

DB is best suitable to use huge.

First lets talk about File concepts.

File

FileWriter

FileReader

BufferedWriter

BufferedReader

PrintWriter

## **File:**

**File f=new File("abc.txt");**

This line 1st checks whether abc.txt file is already available (or) not if it is already available then "f" simply refers that file.  
 If it is not already available then it won't create any physical file just creates a java File object represents name of the file.

Example

-----------

package com.codegnan.fileio;

import java.io.File;

import java.io.IOException;

class FileDemo {

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

// Creating a File object named f

File f = new File("cricket.txt");

// Checking if the file exists

System.out.println(f.exists()); // Output: false

// Creating a new file named "cricket.txt"

f.createNewFile();

// Checking if the file exists after creation

System.out.println(f.exists()); // Output: true

}

}

**A java File object can represent a directory also.**

package com.codegnan.fileio;

import java.io.File;

import java.io.IOException;

import java.io.\*;

class FileDemo {

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

// Creating a File object named f

File f = new File("cricket123");

// Checking if the directory exists

System.out.println(f.exists()); // Output: false

// Creating a new directory named "cricket123"

f.mkdir();

// Checking if the directory exists after creation

System.out.println(f.exists()); // Output: true

}

}

**Note:** in UNIX everything is a file, java "file IO" is based on UNIX operating system hence in java also we can represent both files and directories by File object only.

### **File class constructors:**

1. File f=new File(String name);  
 Creates a java File object that represents name of the file or directory in current working directory.

2. File f=new File(String subdirname,String name);  
 Creates a File object that represents name of the file or directory present in specified sub directory.

3. File f=new File(File subdir,String name);

**Import methods of file class:**

1. **boolean exists();** Returns true if the physical file or directory available.

2. **boolean createNewFile();** This method 1st checks whether the physical file is already available or not if it is already available then this method simply returns false without creating any physical file.  
 If this file is not already available then it will create a new file and returns true

3. **boolean mkdir();** This method 1st checks whether the directory is already available or not if it is already available then this method simply returns false without creating any directory.  
 If this directory is not already available then it will create a new directory and returns true

4. **boolean isFile();** Returns true if the File object represents a physical file.

5. **boolean isDirectory();** Returns true if the File object represents a directory.

6. **String[] list();** It returns the names of all files and subdirectories present in the specified directory.

7. **long length();** Returns the no of characters present in the file.

8. **boolean delete();** To delete a file or directory.

***Requirement:*** Write a program to display the names of all files and directories present in D://javalearning

**package** com.codegnan.fileio;

**import** java.io.File;

**public** **class** ListFilesAndDirectories {

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

// Specify the directory path

String directoryPath = "d://javalearning";

// Create a File object for the directory

File directory = **new** File(directoryPath);

// Ensure that the directory exists

**if** (directory.exists() && directory.isDirectory()) {

// List all files and directories using list()

String[] filesAndDirs = directory.list();

**if** (filesAndDirs != **null**) {

System.***out***.println("Files and directories in " + directoryPath + ":");

**for** (String name : filesAndDirs) {

System.***out***.println(name);

}

}

// Alternatively, list files and directories using listFiles()

File[] fileList = directory.listFiles();

**if** (fileList != **null**) {

System.***out***.println("\nFiles and directories (absolute paths):");

**for** (File file : fileList) {

System.***out***.println(file.getAbsolutePath());

}

}

} **else** {

System.***out***.println("Directory not found or is not a directory.");

}

}

}

## **FileWriter:**

By using FileWriter object we can write character data to the file.

### **Constructors:**

**FileWriter fw=new FileWriter(String name);  
 FileWriter fw=new FileWriter(File f);**

The above 2 constructors meant for overriding.

Instead of overriding if we want append operation then we should go for the following 2 constructors.

**FileWriter fw=new FileWriter(String name,boolean append);  
 FileWriter fw=new FileWriter(File f,boolean append);**

If the specified physical file is not already available then these constructors will create that file.

### **Methods:**

1. **write(int ch);** To write a single character to the file.

2. **write(char[] ch);** To write an array of characters to the file.

3. **write(String s);** To write a String to the file.

4. **flush();** To give the guarantee the total data include last character also written to the file.

5. **close();** To close the stream.

Example

----------

package com.codegnan.fileio;

import java.io.FileWriter;

import java.io.IOException;

public class FileWriterDemo {

public static void main(String[] args) {

// Specify the file name

String fileName = "cricket.txt";

try {

// Create FileWriter object with append mode (true) to append content to file

FileWriter fw = new FileWriter(fileName, true);

// Writing a single character (unicode value of 'c' is 99)

fw.write(99);

// Writing strings

fw.write("codegnan\nsoftware solutions"); // Writing two lines separated by newline character

fw.write("\n"); // Writing an additional newline character

// Writing an array of characters

char[] ch = { 'a', 'b', 'c' };

fw.write(ch); // Writing characters 'a', 'b', 'c'

fw.write("\n"); // Writing an additional newline character

// Flush the writer to ensure all buffered operations are written to the file

fw.flush();

// Close the writer to free up resources

fw.close();

System.out.println("Data has been written to " + fileName);

} catch (IOException e) {

System.out.println("An error occurred while writing to the file: " + e.getMessage());

}

}

}

***Note :***

* The main problem with FileWriter is we have to insert line separator manually , which is difficult to the programmer. ('\n')
* And even line separator varing from system to system.

**FileReader:**

By using FileReader object we can read character data from the file.

**Constructors:**

FileReader fr=new FileReader(String name);  
 FileReader fr=new FileReader (File f);

**Methods:**

1. **int read();** It attempts to read next character from the file and return its Unicode value. If the next character is not available then we will get -1.

int i=fr.read();

System.out.println((char)i);

As this method returns unicode value , while printing we have to perform type casting.

2. **int read(char[] ch);** It attempts to read enough characters from the file into char[] array and returns the no of characters copied from the file into char[] array.

File f=new File("abc.txt");

Char[] ch=new Char[(int)f.length()];

3. **void close();**

Approach 1:

package com.codegnan.fileio;

import java.io.FileReader;

import java.io.IOException;

public class FileReaderDemo {

public static void main(String[] args) {

// Specify the file name to read from

String fileName = "cricket.txt";

try {

// Create a FileReader object

FileReader fr = new FileReader(fileName);

// Read the first character from the file

int i = fr.read();

// Loop until the end of file (EOF) is reached

while (i != -1) {

// Print the character read as a char

System.out.print((char) i);

// Read the next character

i = fr.read();

}

// Close the FileReader to free up resources

fr.close();

} catch (IOException e) {

// Handle exceptions

System.out.println("An error occurred while reading the file: " + e.getMessage());

}

}

}

Output:

Charan

Software solutions

ABC

Approach 2:

**package** com.codegnan.fileio;

**import** java.io.File;

**import** java.io.FileReader;

**import** java.io.IOException;

**public** **class** FileReaderDemo {

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

// Specify the file name to read from

String fileName = "cricket.txt";

**try** {

// Create a File object

File file = **new** File(fileName);

// Create a FileReader object

FileReader fr = **new** FileReader(file);

// Create a char array to store the content of the file

**char**[] buffer = **new** **char**[(**int**) file.length()];

// Read characters into the char array

fr.read(buffer);

// Print the content of the char array

**for** (**char** ch : buffer) {

System.***out***.print(ch);

}

// Close the FileReader to free up resources

fr.close();

} **catch** (IOException e) {

// Handle exceptions

System.***out***.println("An error occurred while reading the file: " + e.getMessage());

}

}

}

**Usage of *FileWriter* and *FileReader* is not recommended because :**

1. While writing data by FileWriter compulsory we should insert line separator(\n) manually which is a bigger headache to the programmer.

2. While reading data by FileReader we have to read character by character instead of line by line which is not convenient to the programmer.

3. To overcome these limitations we should go for BufferedWriter and BufferedReader concepts.

**BufferedWriter:**

By using BufferedWriter object we can write character data to the file.

**Constructors:**

BufferedWriter bw=new BufferedWriter(writer w);  
 BufferedWriter bw=new BufferedWriter(writer w,int buffersize);

**Note:** BufferedWriter never communicates directly with the file it should communicates via some writer object.

**Which of the following declarations are valid?**

1. BufferedWriter bw=new BufferedWriter("cricket.txt"); (invalid)

2. BufferedWriter bw=new BufferedWriter (new File("cricket.txt")); (invalid)

3. BufferedWriter bw=new BufferedWriter (new FileWriter("cricket.txt")); (valid)

**Methods:**

1. write(int ch);

2. write(char[] ch);

3. write(String s);

4. flush();

5. close();

6. newline();  
 Inserting a new line character to the file.

When compared with FileWriter which of the following capability(facility) is available as method in BufferedWriter.

1. Writing data to the file.

2. Closing the writer.

3. Flush the writer.

4. Inserting newline character.

Ans : 4

**package** com.codegnan.fileio;

**import** java.io.BufferedReader;

**import** java.io.BufferedWriter;

**import** java.io.FileReader;

**import** java.io.FileWriter;

**import** java.io.IOException;

**public** **class** BufferedWriterDemo {

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

String fileName = "abc.txt";

**try** {

FileWriter fw = **new** FileWriter(fileName);

BufferedWriter bw = **new** BufferedWriter(fw);

bw.write(100);

bw.newLine();

**char**[] ch = { 'a', 'b', 'c', 'd', 'e' };

bw.write(ch);

bw.newLine();

bw.write("codegnan");

bw.flush();

bw.close();

} **catch** (IOException e) {

e.printStackTrace();

}

System.***out***.println("the content is written to a file" + fileName);

String fileName1 = "abc.txt";

**try** {

FileReader fr = **new** FileReader(fileName1);

BufferedReader br = **new** BufferedReader(fr);

String line = br.readLine();

**while** (line != **null**) {

System.***out***.println(line);

line = br.readLine();

}

} **catch** (IOException e) {

e.printStackTrace();

}

System.***out***.println("Data Has retrived from a file : " + fileName1);

}

}

**BufferedReader:**

This is the most enhanced(better) Reader to read character data from the file.

**Constructors:**

BufferedReader br=new BufferedReader(Reader r);  
 BufferedReader br=new BufferedReader(Reader r,int buffersize);

**Note:** BufferedReader can not communicate directly with the File it should communicate via some Reader object.  
 ***The main advantage of BufferedReader over FileReader is we can read data line by line instead of character by character.***

**Methods:**

1. int read();

2. int read(char[] ch);

3. String readLine();  
 It attempts to read next line and return it , from the File. if the next line is not available then this method returns null.

4. void close();

**package** com.codegnan.fileio;

**import** java.io.BufferedReader;

**import** java.io.BufferedWriter;

**import** java.io.FileReader;

**import** java.io.FileWriter;

**import** java.io.IOException;

**public** **class** BufferedWriterDemo {

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

String fileName1 = "abc.txt";

**try** {

FileReader fr = **new** FileReader(fileName1);

BufferedReader br = **new** BufferedReader(fr);

String line = br.readLine();

**while** (line != **null**) {

System.***out***.println(line);

line = br.readLine();

}

} **catch** (IOException e) {

e.printStackTrace();

}

System.***out***.println("Data Has retrived from a file : " + fileName1);

}

}

## **PrintWriter:**

* This is the most enhanced Writer to write text data to the file.
* By using FileWriter and BufferedWriter we can write only character data to the File but by using PrintWriter we can write any type of data to the File.

### **Constructors:**

PrintWriter pw=new PrintWriter(String name);  
 PrintWriter pw=new PrintWriter(File f);  
 PrintWriter pw=new PrintWriter(Writer w);

PrintWriter can communicate either directly to the File or via some Writer object also.

### **Methods:**

1. write(int ch);

2. write (char[] ch);

3. write(String s);

4. flush();

5. close();

6. print(char ch);

7. print (int i);

8. print (double d);

9. print (boolean b);

10. print (String s);

11. println(char ch);

12. println (int i);

13. println(double d);

14. println(boolean b);

15. println(String s);

**package** com.codegnan.fileio;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**public** **class** PrintWriterDemo {

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

**try** {

PrintWriter out = **new** PrintWriter("abc.txt");

out.write(100);//d

out.println(100);//100

**char**[] ch = { 'a', 'b', 'c' };

out.write(ch);

out.write("codegnan");

out.println("malli");

out.println(**true**);

out.println(20.5);

out.print(**false**);

out.print(65.7f);

out.print("arjun");

out.flush();

out.close();

} **catch** (IOException e) {

e.printStackTrace();

}

System.***out***.println("data is written to a file");

}

}

What is the difference between write(100) and print(100)?  
 In the case of write(100) the corresponding character "d" will be added to the File but in the case of print(100) "100" value will be added directly to the File.  
  
 **Note 1:**

1. The most enhanced Reader to read character data from the File is BufferedReader.

2. The most enhanced Writer to write character data to the File is PrintWriter.

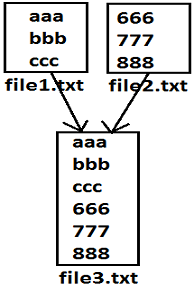
**Note 2:**

1. In general we can use Readers and Writers to handle character data. Where as we can use InputStreams and OutputStreams to handle binary data(like images, audio files, video files etc).

2. We can use OutputStream to write binary data to the File and we can use InputStream to read binary data from the File.

***Write a program to perform File merge(combine) operation.***

Diagram:



**package** com.codegnan.fileio;

**import** java.io.BufferedReader;

**import** java.io.FileReader;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**public** **class** FileWriterDemo1 {

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

// Create file1.txt and write data

**try** (PrintWriter pw1 = **new** PrintWriter(**new** FileWriter("file1.txt"))) {

pw1.println(666);

pw1.println(777);

pw1.println(888);

} **catch** (IOException e) {

System.***out***.println("An error occurred while writing to file1.txt: " + e.getMessage());

}

// Create file2.txt and write data

**try** (PrintWriter pw2 = **new** PrintWriter(**new** FileWriter("file2.txt"))) {

pw2.println("aaa");

pw2.println("bbb");

pw2.println("ccc");

} **catch** (IOException e) {

System.***out***.println("An error occurred while writing to file2.txt: " + e.getMessage());

}

// Merge contents of file1.txt and file2.txt into file3.txt

**try** (BufferedReader br1 = **new** BufferedReader(**new** FileReader("file1.txt"));

BufferedReader br2 = **new** BufferedReader(**new** FileReader("file2.txt"));

PrintWriter pw3 = **new** PrintWriter(**new** FileWriter("file3.txt"))) {

String line;

// Read from file1.txt and write to file3.txt

**while** ((line = br1.readLine()) != **null**) {

pw3.println(line);

}

// Read from file2.txt and write to file3.txt

**while** ((line = br2.readLine()) != **null**) {

pw3.println(line);

}

System.***out***.println("Merged contents of file1.txt and file2.txt into file3.txt.");

} **catch** (IOException e) {

System.***out***.println("An error occurred while merging files: " + e.getMessage());

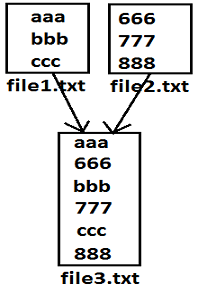
}

}

}

***Requirement***: Write a program to perform file merge operation where merging should be performed line by line alternatively.

Diagram



**package** com.codegnan.fileio;

**import** java.io.BufferedReader;

**import** java.io.FileNotFoundException;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**public** **class** MergeFilesDemo {

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

**try** {

BufferedReader br1 = **new** BufferedReader(**new** FileReader("file1.txt"));

BufferedReader br2 = **new** BufferedReader(**new** FileReader("file2.txt"));

PrintWriter pw = **new** PrintWriter("file3.txt");

String line1 = br1.readLine();

String line2 = br2.readLine();

**while** (line1 != **null** || line2 != **null**) {

**if** (line1 != **null**) {

pw.println(line1);

line1 = br1.readLine();

}

**if** (line2 != **null**) {

pw.println(line2);

line2 = br2.readLine();

}

}

pw.flush();

pw.close();

br1.close();

br2.close();

} **catch** (FileNotFoundException e) {

e.printStackTrace();

}

}

}