```
File(IO Package)
==============
  Agenda:
1. File
2. FileWriter
3. FileReader
4. BufferedWriter
5. BufferedReader

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:
import java.io.*;
class FileDemo{
      public static void main(String[] args)throws IOException{
            File f=new File("abc.txt");
            System.out.println(f.exists());//false

            f.createNewFile();
            System.out.println(f.exists());//true
      }
}
1st run
=======
false
true

2nd run
======
true
true

=> A java File object can represent a directory also.
Example:
import   java.io.File;
import   java.io.IOException;

class FileDemo{
  public static void main(String[] args)throws IOException{
      File f=new File("cricket123");
      System.out.println(f.exists());//false

      f.mkdir();//Creates a new directory
      System.out.println(f.exists());//true
 }
}
1st run
=======
false
true

2nd run
======
true
```

```
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.


Constructors of File class
==========================
 File f=new File(String fname)
 File f=new File(String directoryName,String fileName);
 File f=new File(File f,String fileName);

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.
            eg#1. File f=new File("abc.txt");

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.
            eg#1. File f1=new File("abc");
                        f1.mkdir();
                           File f2=new File("abc","demo.txt");
                        f2.createNewFile();

3. File f=new File(File subdir,String name);
            eg#1.File f1=new File("abc");
                        f1.mkdir();
                      File f2=new File(f1,"demo.txt");
                        f2.createNewFile();

Requirement
===========
=> Write code to create a file named with demo.txt in current working directory.
            cwd
             |=> abc.txt

Program:
import java.io.*;
class FileDemo{
      public static void main(String[] args)throws IOException{
            File f=new File("demo.txt");
            f.createNewFile();
      }
}
Requirement
=> Write code to create a directory named with IPLTeam in current working directory
and create a  file named with abc.txt in that directory.
             cwd
               |=> IPLTeam
                      |=> abc.txt
Program:
import java.io.*;
class FileDemo{
  public static void main(String[] args)throws IOException{
      File f1=new File("IPLTeam");
      f1.mkdir();
      File f2=new File("IPLTeam","abc.txt");
      f2.createNewFile();
  }
}
```

Requirement: Write code to create a file named with rcb.txt present in D:\IPLTeam folder.

```
                  D
                  |=> IplTeam
                        |-> rcb.txt
```

Program:

```
import java.io.*;
class FileDemo{
      public static void main(String[] args)throws IOException{
            File f=new File("D:\\IPLTeam","rcb.txt");
            f.createNewFile();
      }
}
```

Assuming C:\\IPLTeam should be already available otherwise it would result in "FileNotFoundException".

Important methods of File class
===============================
1. boolean exists();
2. boolean createNewFile()
3. boolean mkdir()
4. boolean isFile();
5. boolean isDirectory()
6. String[] list();
7. long length();
8. boolean delete()

Important 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:\EnterpriseJava
Requirement: Write a program to display only file names.
Requirement: Write a program to display only directory names.

Requirement: Write a program to display the names of all files and directories present
           in D:\EnterpriseJava

```
import java.io.File;
import java.io.IOException;

public class TestApp {
      public static void main(String[] args)throws IOException {
            File f=new File("D:\EnterpriseJava");
            String[] s= f.list();
            int count=0;
            for(String s1:s){
                  count++;
                  System.out.println(s1);
            }
            System.out.println("The no of files are :: "+count);
      }
}
```

Requirement: Write a program to display only file names.

```
import java.io.File;
import java.io.IOException;

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

            File f=new File("D:\EnterpriseJava");
            String[] s= f.list();
            int count=0;

            for(String s1:s){
                  File f1=new File(f,s1);
                  if (f1.isFile()){
                        count++;
                        System.out.println(s1);
                  }
            }
            System.out.println("The no of Directories are :: "+count);
      }
}
```

Requirement: Write a program to display only directory names

```
import java.io.File;
import java.io.IOException;

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

            File f=new File("D:\EnterpriseJava");
            String[] s= f.list();
            int count=0;

            for(String s1:s){
                  File f1=new File(f,s1);
                  if (f1.isDirectory()){
                        count++;
                        System.out.println(s1);
                  }
            }
            System.out.println("The no of Directories are :: "+count);
      }
}
```

FileWriter
========
Constructors:

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

The above 2 constructors meant for overriding the data to the file.

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.

eg#1.

```
import java.io.FileWriter;
import java.io.IOException;

public class TestApp {
      public static void main(String[] args)throws IOException {
                  FileWriter fw=new FileWriter("abc.txt");
                  fw.write(73);
                  fw.write("neuron\nTechnology\nPrivate\nLimited");
                  fw.write("\n");
                  char ch[] ={'a','b','c'};
                  fw.write(ch);
                  fw.flush();
                  fw.close();
      }
}
```

A new file will be created automatically

```
abc.txt
=======
Ineuron
Technology
Private
Limited
abc
```

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.
 =>Represenation of "\n" would vary 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.

2. int i=fr.read();

3. System.out.println((char)i);
      As this method returns unicodevalue , while printing we have to perform type casting.

4. 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.

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

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

7. void close();

```
eg#1.
import java.io.FileReader;
import java.io.IOException;

public class TestApp {
      public static void main(String[] args)throws IOException {
                  FileReader fr=new FileReader("abc.txt");
                  int i=fr.read();
                  while(i!=-1){
                        System.out.println((char)i);
                        i=fr.read();
                  }
      }
}
```

eg#2. Reading an array of characters

```
abc.txt
=======
  1000 characters are available

Scenario1:
       FileReader fr=new FileReader("abc.txt");
       char[] ch=new char[10];
       int noOfCharactersCopied=fr.read(ch);

Scenario2:
       FileReader fr=new FileReader("abc.txt");
       char[] ch=new char[10000];
       int noOfCharactersCopied=fr.read(ch);


import java.io.FileReader;
import java.io.IOException;
import java.io.File;

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

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

                  FileReader fr=new FileReader(f);
                  char ch[] = new char[(int)f.length()];

                  fr.read(ch);

                  String data=new String(ch);
                  System.out.println(data);

                  fr.close();
      }
}
```

Usage of FileWriter and FileReader is not recommended because of following reason

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.

Assume we need to search for a 10 digit mobile no present in a file called "mobile.txt"
  =>Since we can read only character just to search one mobile no 10 searching and to search 10,000 mobile no we need to read 1cr times,
      so performance is very low.

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

BufferedWriter:
    It can't communicate with the file directly, it can communicate only with writer Object.

Constructor

```
   BufferedWriter bw=new BufferedWriter(Writer w);
   BufferedWriter bw=new BufferedWriter(Writer w,int buffersize);
```

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
4. BufferedWriter bw=new BufferedWriter(new BufferedWriter(new FileWriter("crickter.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.
Answer: 4(newLine())

```
import java.io.*;
class TestApp
{
      public static void main(String[] args)throws IOException
      {
            BufferedWriter bw = new BufferedWriter(new FileWriter("abc.txt"));
            bw.write(73);
            bw.write("neuron");
            bw.newLine();
            bw.write("technology");
            bw.newLine();
            char ch[] = {'a','b','c'};
            bw.write(ch);


            bw.flush();

            bw.close();

      }
}
```

Note
1.bw.close()// recomended to use
2.fw.close()// not recomended to use
3.bw.close()// not recomended to use
  fw.close()
=> When ever we are closing BufferedWriter automatically underlying writer will be closed and we are not close explicitly.

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();

eg#1.Read the data from the file called "abc.txt"

```
import java.io.FileReader;
import java.io.IOException;
import java.io.BufferedReader;
public class TestApp {
      public static void main(String[] args)throws IOException {

                  FileReader fr=new FileReader("abc.txt");
                  BufferedReader br=new BufferedReader(fr);
                  String line= br.readLine();
                  while(line!=null){
                        System.out.println(data);
                        line=br.readLine();
                  }
                  br.close();
      }
}
```

Note:
1.br.close()// recomended to use
2.fw.close()// not recomended to use
3.br.close()// not recomended to use
   fw.close()

=> Whenever we are closing BufferedReader automatically underlying FileReader will be closed it is not required to close explicitly.
=> Even this rule is applicable for BufferedWriter also.

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);
```

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);
```

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

```
      Character Data => Reader and Writer
      Binary Data       => InputStream and OutputStream
```

Note 3:

```
                                                            Object
                                                              |
                  ==============================================
                  |
       |
                    Writer(AbstractClass)
Reader(AbstractClass)
                     |
                   |

==================================================================================
          |                               |                            |
                           |                              |

 OutputStreamWriter BufferedWriter PrintWriter            InputStreamReader
BufferedReader
          |
                              |
      FileWriter
               FileReader
```

Requirement => file1.txt ,file2.txt copy all the contents to file3.txt

```
import java.io.*;
class TestApp {
      public static void main(String[] args)throws IOException {
            PrintWriter pw =new PrintWriter("file3.txt");

            //copy from file1.txt to file3.txt
            BufferedReader br=new BufferedReader(new FileReader("file1.txt"));
            String line = br.readLine();
            while(line!=null){
                  pw.println(line);
                  line = br.readLine();
            }

            //copy from file2.txt to file3.txt
            br=new BufferedReader(new FileReader("file2.txt"));
            line = br.readLine();
            while(line!=null){
                  pw.println(line);
                  line = br.readLine();
            }

            //closing the resources
            pw.flush();
            br.close();
            pw.close();


      }
}
```

Requirement => file1.txt file2.txt copy one line from file1.txt and from file2.txt to file3.txt.

```
import java.io.*;

class TestApp
{
      public static void main(String[] args)throws IOException
      {
            PrintWriter pw =new PrintWriter("file3.txt");

            //copy from file1.txt to file3.txt
            BufferedReader br1=new BufferedReader(new FileReader("file1.txt"));
            BufferedReader br2=new BufferedReader(new FileReader("file2.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();
                  }
            }

            //closing the resources
            pw.flush();
            br1.close();
            br2.close();
            pw.close();


      }
}
```

Requirement => Write a program to perform extraction of mobile no only if there is no duplicates

```
import java.io.*;

class TestApp
{
      public static void main(String[] args)throws IOException
      {
            PrintWriter pw =new PrintWriter("output.txt");

            //copy from file1.txt to file3.txt
            BufferedReader br1 =new BufferedReader(new FileReader("input.txt"));
            String line = br1.readLine();

            BufferedReader br2 =null;

            while(line!=null)
            {
                  boolean isAvailable = false;
                   br2=new BufferedReader(new FileReader("delete.txt"));
                  String target = br2.readLine();

                  while(target!=null)
                  {
                        if(line.equals(target))
                        {
                              isAvailable = true;
                              break;
                        }
                        target = br2.readLine();
                  }
                  if (isAvailable==false)
                  {
                        pw.println(line);
                        pw.flush();//flush to ensure all data is written to the
file
                  }

                  line = br1.readLine();
            }
            //closing the resources
            br1.close();
            br2.close();
            pw.close();


      }
}
```

Requirement => Write a program to remove duplicates from the file

```
import java.io.*;

class TestApp
{
      public static void main(String[] args)throws IOException
      {
            PrintWriter pw =new PrintWriter("output.txt");

            //copy from file1.txt to file3.txt
            BufferedReader br1 =new BufferedReader(new FileReader("input.txt"));
            String line = br1.readLine();

            BufferedReader br2 =null;

            while(line!=null)
            {
                  boolean isAvailable =false;
                  br2 =new BufferedReader(new FileReader("output.txt"));
                  String target = br2.readLine();

                  while(target!=null)
                  {
                        if (line.equals(target))
                        {
                              isAvailable = true;
                              break;
                        }
                        target = br2.readLine();
                  }

                  if(isAvailable==false){
                        pw.println(line);
                        pw.flush();
                  }

                  line = br1.readLine();
            }
            //closing the resources
            br1.close();
            br2.close();
            pw.close();


        }
}
```