The io package supports Java's basic I/O system.

## File:

- The File class does not specify how information is retrieved from or stored in files.
- It describes the properties of a file itself.
- 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
- Files are a primary source and destination for data within many programs
- The following constructors can be used to create File objects:

  File(String directoryPath)
  File(String directoryPath, String filename)
  File(File dirObj, String filename)
  File(URI uriObj)

## Directories:

- A directory is a File that contains a list of other files and directories.
- When you create a File object that is a directory.

## The Autocloseable, Closeable, and Flushable Interfaces

Closeable and Flushable: They are defined in java.io and were added by JDK 5.

AutoCloseable: was added by JDK 7. It is packaged in java.lang.

- AutoCloseable provides support for the try-with-resources statement, which automates the process of closing a resource.
- Autocloseable is called automatically at the end of a try-with-resources statement, thus eliminating the need to explicitly call **close().**
- The AutoCloseable interface defines only the **close( )** method
- This method closes the invoking object, releasing any resources that it may hold.
- The Closeable interface also defines the **close( )** method. Objects of a class that implement Closeable can be closed.
- Any class that implements Closeable also implements AutoCloseable.
- Objects of a class that implements Flushable can force buffered output to be written to the stream to which the object is attached. It defines the **flush( )** method

## I/O Exceptions:

**(i) IOException:**

- if an I/O error occurs, an IOException is thrown.

**(ii) FileNotFoundException:**

- if a file cannot be opened, a FileNotFoundException is thrown.
- FileNotFoundException is a subclass of IOException, so both can be caught with a single catch that catches IOException.

## The stream classes:

**Stream:** A stream represents a flow of data.

1. Byte Streams: provides a convenient means for handling input and output of bytes.
   i. InputStream: to read bytes from a file
   ii. OutputStream: to write bytes in a file

2. Character Streams: a convenient means for handling input and output of characters.
   I. Reader: to read characters from a file
   ii. Writer: to write characters in a file

# A. The Byte Streams:

## FileInputStream:

- The FileInputStream class creates an InputStream that you can use to read bytes from a file.
- Two commonly used constructors are shown here:
  FileInputStream(String filePath)
  FileInputStream(File fileObj)
- Here, filePath is the full path name of a file, and fileObj is a File object that describes the file.
- Either can throw a FileNotFoundException.

**FileInputStream abc = new FileInputStream("text.txt");**

## FileOutputStream:

- FileOutputStream creates an OutputStream that you can use to write bytes to a file.
- It implements the AutoCloseable, Closeable, and Flushable interfaces.

- Four of its constructors are shown here:
  FileOutputStream(String filePath)
  FileOutputStream(File fileObj)
  FileOutputStream(String filePath, boolean append)
  FileOutputStream(File fileObj, boolean append)
- Either can throw a FileNotFoundException.

**FileOutputStream xyz = new FileOutputStream("abc.txt");**

## PrintStream:

- Output stream that contains **print()** and **println()**

## DataOutputStream:

- Output stream that contains methods for writing the java standard data types

## DataInputStream:

- Input stream that contains methods for reading the java standard data types

## RandomAccessFile:

- RandomAccessFile is special because it supports positioning requests—that is, you can position the file pointer within the file.

# B. The Character Streams:

## FileWriter:

- output stream that writes to a file
  **FileWriter ab = new FileWriter("file1.txt");**

## FileReader:

- input stream that reads from a file

**FileReader xy = new FileReader("file1.txt");**

## PrintWriter:

- Output stream that contains **print()** and **println()**

## The Predefined Streams:

- **System** contains three predefined stream variables: **in**, **out**, and **err**.
- **System.in** is an object of type **InputStream**; **System.out** and **System.err** are objects of type **PrintStream**.

## The Console Class:

- It is used to read from and write to the console

## Reading Console Input:

- In Java, console input is accomplished by reading from **System.in**
- To obtain a character- based stream that is attached to the console, wrap **System.in** in a **BufferedReader** object.
- **BufferedReader** supports a buffered input stream: **InputStreamReader**, which converts bytes to characters.

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

## Reading Characters:

```
import java.io.*;
class readch
{
	public static void main(String args[]) throws IOException
	{
		char ch;
		  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		  System.out.println("Enter characters and press 'q' to quit.");
		 // read characters
		do{
			ch =  (char) br.read();
			System.out.println(ch);
		    } while(ch != 'q');
	}
}
```

## Reading Strings:

```
import java.io.*;
class brreadLines
{
        public static void main(String args[]) throws IOException
    {
                // create a BufferedReader using System.in
               BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
               String str;
               System.out.println("Enter lines of text.");
               System.out.println("Enter 'stop' to quit.");
               do{
                        str = br.readLine();
                        System.out.println(str);
                  } while(!str.equals("stop"));
    }
}
```

## Text Editor:

```
import java.io.*;
class tinyEdit
{
        public static void main(String args[]) throws IOException
    {
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                String str[] = new String[4];
                System.out.println("Enter lines of text & enter 'stop' to quit.");
                for(int i=0; i<4; i++)
                {
                        str[i] = br.readLine();
                        if(str[i].equals("stop")) break;
                }
               System.out.println("\nHere is your file:");     // display the lines
               for(int i=0; i<4; i++)
               {
                        if(str[i].equals("stop")) break;
                        System.out.println(str[i]);
               }
    }
}
```

## Writing Console Output:

- Console output is most easily accomplished with **print( )** and **println( )**
- These methods are defined by the class **PrintStream** (which is the type of object referenced by **System.out**)
- Even though **System.out** is a byte stream, using it for simple program output is still acceptable.

## The PrintWriter Class:

- **PrintWriter** is one of the character-based classes. Using a character-based class for console output makes internationalizing your program easier.

**PrintWriter (OutputStream *outputStream*, boolean *flushingOn*)**

- Here, *outputStream* is an object of type **OutputStream**, and *flushingOn* controls whether Java flushes the output stream every time a **println( )** method (among others) is called. If *flushingOn* is **true**, flushing automatically takes place. If **false**, flushing is not automatic.
- **PrintWriter** supports the **print( )** and **println( )** methods.

```
import java.io.*;
public class printwriterDemo
{
    public static void main(String args[])
    {
            PrintWriter pw = new PrintWriter(System.out,true);
            String s = "This is a string.";
            int i = 154;
            double d = 409754.67;
            pw.println(s+" "+i+" "+d);
    }
 }
```

## Serialization:

Serialization is the process of writing the state of an object to a byte stream. This is useful when you want to save the state of your program to a persistent storage area, such as a file. At a later time, you may restore these objects by using the process of deserialization.

```
import java.io.*;

class Student implements Serializable

{

        int id;     String name;

        public Student(int id, String name)

        {

                this.id = id;

                this.name = name;

        }

}

class serializationdemo

{

        public static void main(String args[])throws Exception

        {

                Student s1 =new Student(211,"ravi");

                FileOutputStream fout= new FileOutputStream("f.txt");

                ObjectOutputStream out= new ObjectOutputStream(fout);

                out.writeObject(s1);

                out.flush();

                System.out.println("success");

        }

}
```

**Writing, Reading and Displaying contents of a File:**

```
import java.io.*;
class read_write
{
	public static void main(String args[])
	{
		// to write content in file
		try
		{
			FileOutputStream fout=new FileOutputStream("info.dat");
			String country="Nepal";
			String city ="Kathmandu";
			byte bcountry[]=country.getBytes();	//converting string into byte array
			byte bcity[]=city.getBytes();
			fout.write(bcountry);
			fout.write(bcity);
			fout.close();
		}catch(Exception e)
		{
			System.out.println(e);
		}

		//	to read and display content of file
		try
		{
			FileInputStream fin=new FileInputStream("info.dat");
			int i=0;
			while((i=fin.read())!=-1)
			{
				System.out.println((char)i);
			}
			fin.close();
		}catch(Exception e)
		{
			System.out.println(e);
		}
	}
}
```

**Copy one File to another:**

```
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 length;
			while ((length = in.read()) != -1)
			{
				out.write(length);
			}
		}
		finally
		{
			if (in != null)
			{
				in.close();
			}
			if (out != null)
			{
				out.close();
			}
			System.out.println("File Copied Successfully.");
		}
	}
}
```

**Copy one File to another(user input):**

```
import java.io.*;
import java.util.Scanner;
public class Copy_User
{
	public static void main(String args[]) throws IOException
	{
		String src, dest;
		Scanner scan = new Scanner(System.in);
		System.out.print("Enter Source File Name (with extension): ");
		src = scan.nextLine();
		System.out.print("Enter Destination File Name (with extension): ");
		dest = scan.nextLine();

		FileInputStream in = null;
		FileOutputStream out = null;
		try
		{
			in = new FileInputStream(src);
			out = new FileOutputStream(dest);

			int length;
			while ((length = in.read()) > 0)
			{
				out.write(length);
			}
		}
		finally
		{
			if (in != null)
				in.close();
			if (out != null)
				out.close();
			System.out.println("File Copied..");
		}

	}
}
```

```
//WAP that displays all the Read only Files of a given Folder
import java.io.*;
class FilesListFromFolder
{
   public static void main(String a[])
   {
      File file = new File("D:/MyFolder/");
      file.setReadOnly();
      File[] files = file.listFiles();
      for(File f: files)
      {
         System.out.println(f.getName());
      }
   }
}
```

**/*Write a program to take employee id, name and DOB from user and store it in a file "Emp.txt"and then display the content of "Emp.txt"*/**

```
import java.io.*;
import java.util.Scanner;
class fil
{
        public static void main(String[]args)throws IOException
        {
                try
                {
                        FileWriter fw=new FileWriter("Emp.txt");
                        BufferedWriter bw= new BufferedWriter(fw);
                        String name,id,DOB;
                        Scanner scan=new Scanner(System.in);
                        System.out.print("Enter name,id and Dob: ");
                        name=scan.nextLine();
                        id=scan.nextLine();
                        DOB=scan.nextLine();

                        bw.write(id);
                        bw.write(DOB);
                        bw.write(name);
                        bw.close();
                }catch(Exception e)
                {
                        System.out.println(e);
                }
```

```
		try
		{
			FileReader fr = new FileReader("Emp.txt");
			BufferedReader br = new BufferedReader(fr);
			String msg=null;
			while((msg=br.readLine())!=null)
			{
				System.out.println(msg);
			}
			br.close();
		}catch(Exception e)
		{
			System.out.println(e);
		}
	}
}
```

## // WAP that displays content of folder database stored in d: drive

```
import java.io.*;
class ContentFromFolder
{
	public static void main(String a[])
	{
		String dirname = "D:/database"
		File f1 = new File(dirname);
		if(f1.isDirectory())
		{
			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");
			}
			file.listFiles();
			for(File f: files)
			{
				System.out.println(f.getName());
			}
		}
	}
}
```

## /* Program that reads line of text from keyboard and write to file. Also read the content of the same file and display on monitor */

```
import java.io.*;
class read_write_keyboard
{
	public static void main(String args[]) throws IOException
	{
		BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		String str;
		FileWriter fw = null;
		System.out.println("Enter lines of text.");
		System.out.println("Enter 'stop' to quit.");
		try{
			fw = new FileWriter("hello.txt");
			do{
				str = br.readLine();
				System.out.println(str);
				fw.write(str);
			    } while(!str.equals("stop"));
		    }catch(IOException e)
		    {
			System.out.println("IO Interruption");
		    }
	    }
}
```