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. <u>The Byte Streams:</u>

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”);

PrinWriter:

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

The Console Class:

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

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 and 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:

```
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");
        }
}
```

Display only files from a folder

```
import java.io.*;

public class folder_display

{

	public static void main(String [] args)

	{

		File directory = new File("D:/Database");

		String [] list = directory.list();

		for(int i=0;i<list.length;i++)

		{

			File f = new File("D:/Database",list[i]);

			if(f.isFile())

				System.out.println((i+1)+" : "+list[i]);

		}

	}

}
```