Website

https://www.curiousinfotechsolutions.com

Managing Input/Output Files in Java

Java Programming

Day-20

Email id:
info@curiousinfotechsolutions.com

Syllabus Day-20

- **Brief**
  - ✓**What is a file in Java?**
  - ✓**Concept of Streams**
  - ✓Using Input and Output Stream
  - ✓**Stream Classes**
  - ✓**Classification of Java Stream Classes**
  - ✓**Using the File Class**
  - ✓**Input/Output Exceptions**
  - ✓**Common Stream Classes used for I/O Operations**
  - ✓**Examples: Writing Bytes, Reading Bytes, Reading/Writing Characters**
  - ✓**Examples:** Reading / Writing primitive data, Concatenating and Buffering Files

File..Introduction

- So far, we have used variables and arrays for storing data inside the programs. This approach poses the following problems:
  - The data is lost either when a variable goes out of scope or when the program is terminated. That is, the storage is temporary.
  - It is difficult to handle large volumes of data *using variables and arrays*.
- We can **overcome these problems** by storing data on secondary storage devices. The data is store in these devices using the concept of files.

Contd..File Intro

- A file is a collection of related records placed in a *particular area on the disk*. A record is composed of several fields and a field is a group of characters.
- **Characters in Java are Unicode characters** composed of two bytes, each byte containing eight binary digits, 1 or 0.
- **Storing and managing data using files** I known as file processing which includes tasks such as creating files, updating files and manipulation of data.

Contd..File Intro

- Java supports many powerful features for managing input and output of data using files.
- Reading and writing of data in a file can be done at the level of bytes or characters or fields depending on the requirements of a particular applications.
- Java also provides capabilities to read and write class objects directly.
- The process of reading and writing objects is called object **serialization**.

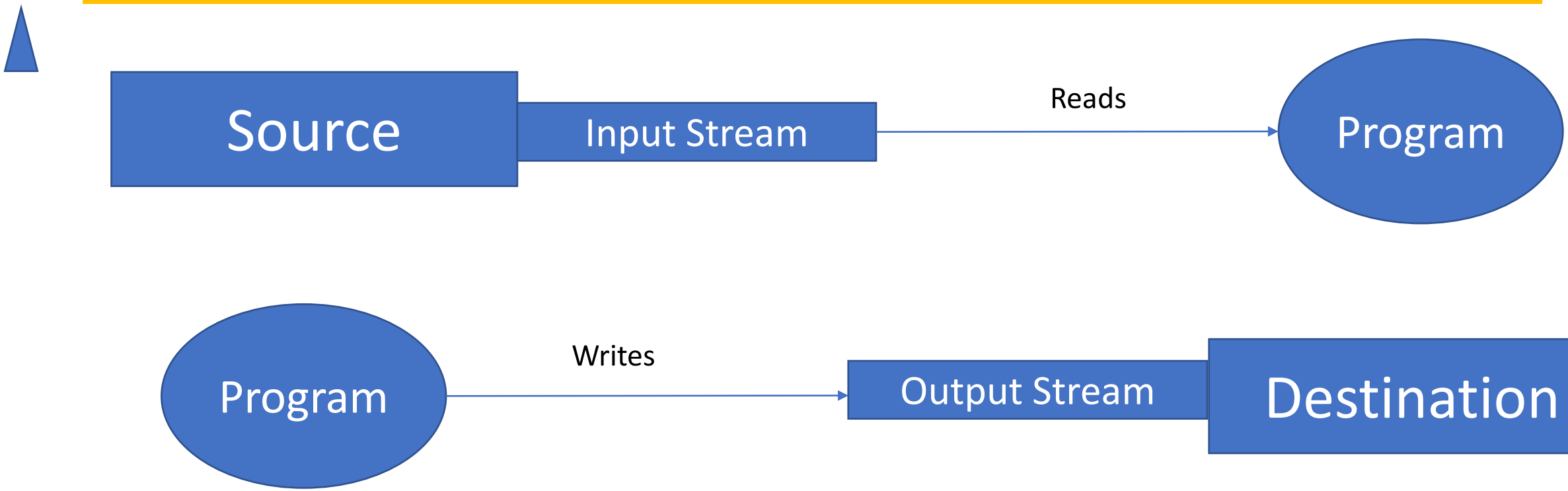
Concept of Streams

- In file processing, input refers to the flow of data into a program and output *means the flow of data out of a program*.
- Input to a program may come from the keyboard, the mouse, the memory, the disk, a network, or another program.
- Similarly, output from a program may go to the screen, the printer, the memory, the disk, a network, or another program.
- Java uses the concept of streams *to represent the ordered sequence of data*, a common characteristic shared by all the input/output devices.

Contd.. Concept of Streams

- A stream in Java is a path along which ***data flows***.
- It has a **source of data** and a **destination for the data**.
- Both **the source and the destination** may be physical devices or programs or other streams in the same program.
- The concept of sending data from one **stream** to another has made streams in Java a powerful tool for file processing.
- For Example: we can use **one stream to get raw data in binary format** and then use another stream in series to **convert it to integers**.

Using Input and Output Stream



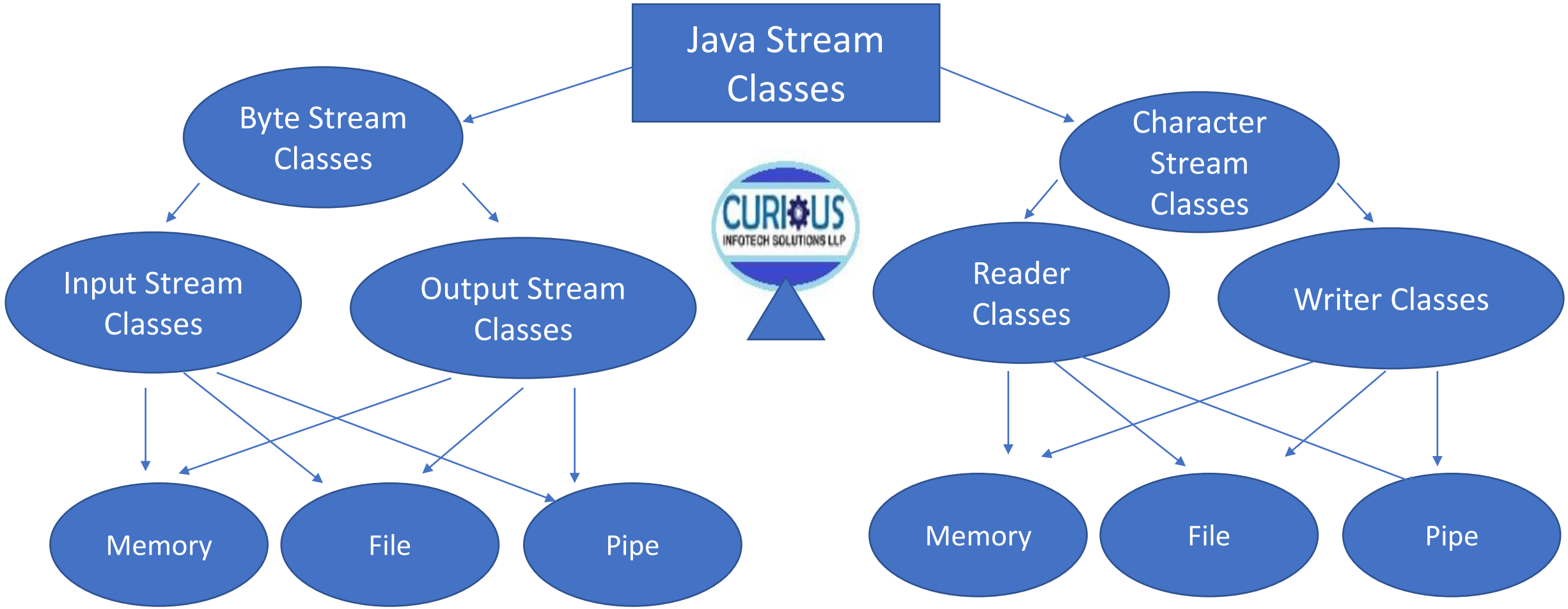
CURIOUS INFOTECH SOLUTIONS LLP

https://www.curiousinfotechsolutions.com

Stream Classes

- The java.io package contains a large number of stream classes that provide capabilities for processing all types of data.
- These classes may be categorized into two groups based on the data type on which they operate:
  1. Byte stream classes that provide support for handling I/O *operations on bytes*.
  2. Characters stream classes that provide support for managing I/O *operations on characters*.

Classification of Java Stream Classes



Using the File Class

- The java.io package includes a class known as the File class that provides support for creating files and directories. The class includes several constructors for instantiating the File objects.
- This class also contains several methods for supporting the operations such as:
  - Creating a file, Opening a file, Closing a file, Deleting a file
  - Getting the ***name of a file***, Getting the ***size of a file***
  - Checking the existence of a file, Renaming a file
  - Checking whether the ***file is writable***, Checking whether the ***file is readable***

Input/Output Exceptions

I/O Exception Class	Function
EOFException	Signals that an end of the file or end of stream has been reached unexpectedly during Input.
FileNotFoundException	Inform that file could not be found
InterruptedIOException	Warns that an I/O operations has been interrupted
IOException	Signals that an I/O exception of some sort has occurred.

Contd.. Input/Output Exceptions

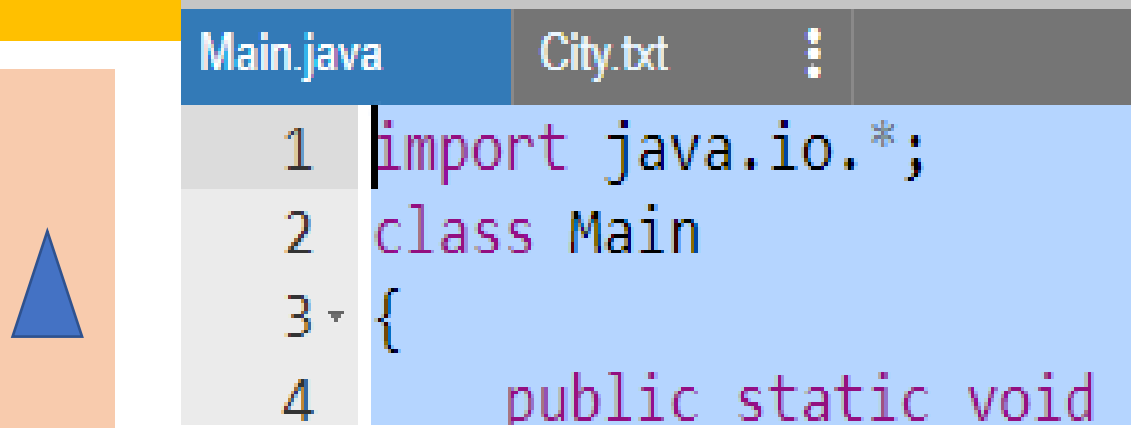
- try
- {
  ….. // I/O Statement
- }
- catch(IOException e)
- {
  ….// Message output Statement
- }

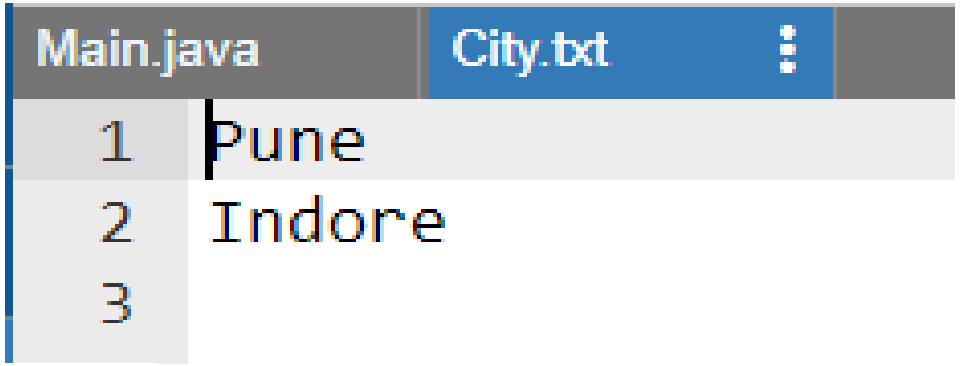
Common Stream Classes used for I/O Operations

Source or Destinations	Characters		Bytes	
	Read	Write	Read	Write
Memory	CharArrayReader	CharArrayWriter	ByteArrayInputStream	ByteArrayOutputStream
File	FileReader	FileWriter	FileInputStream	FileOutputStream
Pipe	PipedReader	PipedWriter	PipedInputStream	PipedOuputStream

Writing Bytes


```
import java.io.*;
class Main
{
   public static void main(String arg[])
   {
      byte cities[] ={'P','u','n','e','\n','I','n','d','o','r', 'e','\n'};
      FileOutputStream outfile = null;
      try{
      outfile = new FileOutputStream("City.txt");
      outfile.write(cities);
      outfile.close();
      }
      catch(IOException ioe)
      {
         System.out.println(ioe);
         System.exit(-1);
      }
   }
}
```





Reading Bytes

Main.java | City.txt

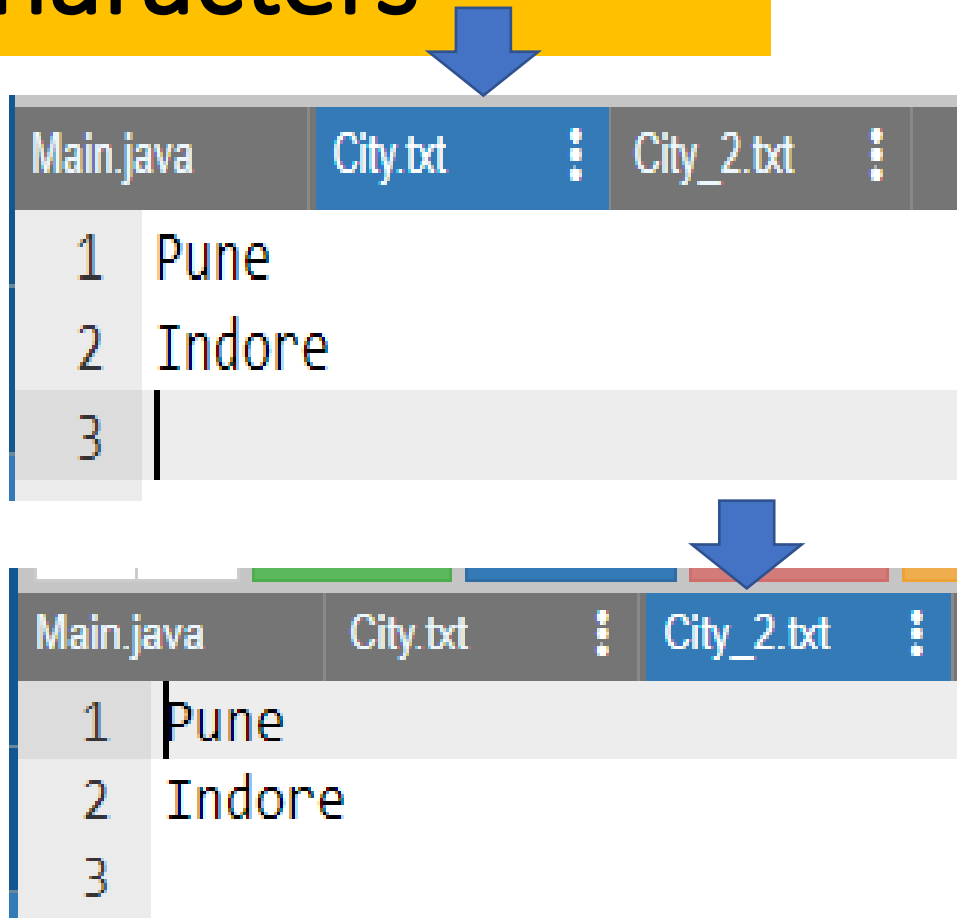
```
1  Pune
2  Indore
3
```

```
Pune
Indore


...Program finished with exit code 0
Press ENTER to exit console.
```

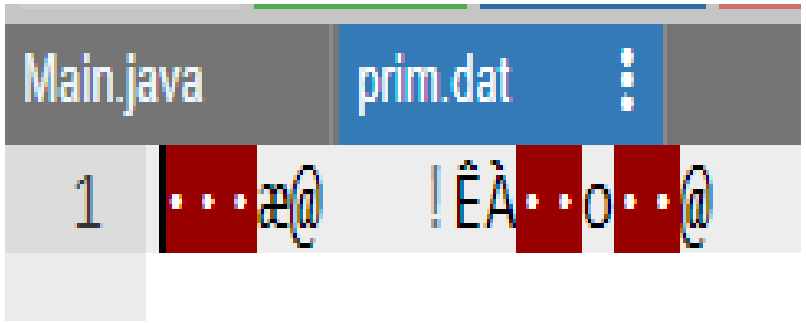
```
import java.io.*;
class Main
{
   public static void main(String arg[])
   {
      int b;
      FileInputStream infile = null;
      try{
      infile = new FileInputStream("City.txt");
      while((b=infile.read()) != -1)
      {
         System.out.print((char)b);
      }
      infile.close();
      }
      catch(IOException ioe)
      {
         System.out.println(ioe);
         System.exit(-1);
      }
   }
}
```

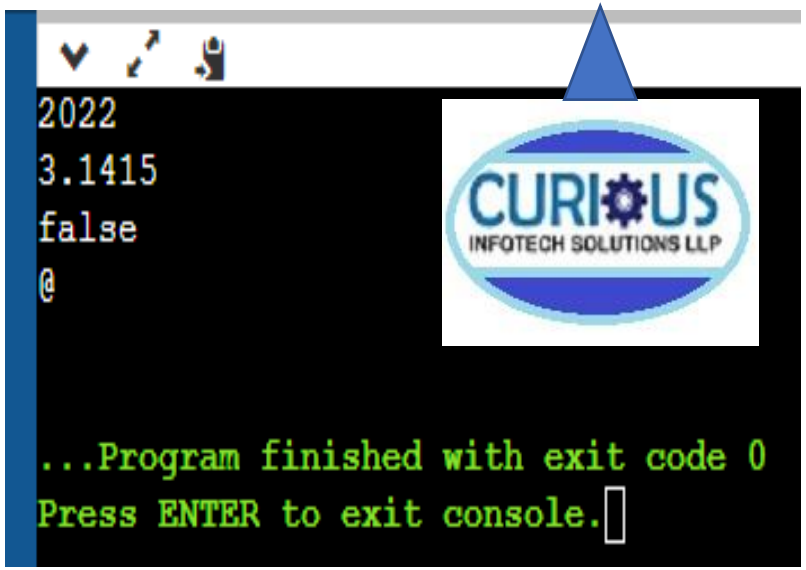
Reading/Writing Characters




```
import java.io.*;
class Main
{
   public static void main(String arg[])
   {
      int b;
      File inFile = new File("City.txt");//Source File
      File outFile = new File("City_2.txt");//Target File
      FileReader ins = null; FileWriter outs = null;
      try{
         ins = new FileReader(inFile);
         outs = new FileWriter(outFile);
         int ch;
         while ((ch=ins.read()) != -1) { outs.write(ch); }
         ins.close(); outs.close();
      }
      catch(IOException e)
      {
         System.out.println(e);
         System.exit(-1);
      }
   }
}
```

Reading / Writing primitive data






```
import java.io.*;
class Main
{
    public static void main(String arg[]) throws IOException
    {
        File primitive = new File("prim.dat");
        FileOutputStream fos = new FileOutputStream(primitive);
        DataOutputStream dos = new DataOutputStream(fos);
        //Writing primitive data to the prim.dat file
        dos.writeInt(2022);
        dos.writeDouble(3.1415);
        dos.writeBoolean(false);
        dos.writeChar('@');
        dos.close(); fos.close();
        //Reading primitive data from the prim.dat file
        FileInputStream fis = new FileInputStream(primitive);
        DataInputStream dis = new DataInputStream(fis);
        System.out.println(dis.readInt());
        System.out.println(dis.readDouble());
        System.out.println(dis.readBoolean());
        System.out.println(dis.readChar());
        dis.close(); fis.close();
    }
}
```

Concatenating and Buffering Files

https://www.curiousinfotechsolutions.com

```
import java.io.*;
class Main
{
   public static void main(String arg[]) throws IOException
   {
      //Declare file Stream
      FileInputStream file1 = null;
      FileInputStream file2 = null;
      //Declare file3 to store combined Files
      SequenceInputStream file3 = null;
      //Open the files to be concatenated
      file1 = new FileInputStream("Text1.txt");//Soucre File-1
      file2 = new FileInputStream("Text2.txt");//Source File-2
      //concatenate file1 and file2 into file3
      file3 = new SequenceInputStream(file1,file2);
      //Create Buffered Input and Output streams
      BufferedInputStream inBuffer = new BufferedInputStream(file3);
      BufferedOutputStream outBuffer = new BufferedOutputStream(System.out);
```

```
//Read and Write till the end of BufferedOutputStream
      int ch;
      while ((ch = inBuffer.read()) != -1)
      {
         outBuffer.write((char)ch);
      }
      inBuffer.close();
      outBuffer.close();
      file1.close();
      file2.close();
   }
}
```