UNIT – VI

FILE HANDLING

6.1 Stream Classes

- **Stream** :

- Java programs perform I/O through streams.
- Streams are pipelines for sending and receiving information from/to Java programs.
- A stream is linked to a physical device by the Java I/O system.
- When a stream of data is being sent or received, it is referred as writing and reading a stream.
- Three streams are created for us automatically:
    1. System.out: standard output stream
    2. System.in: standard input stream
    3. System.err: standard error

6.1 Stream Classes

- Streams are also classified in to two categories

    1. Character streams
    2. Byte streams

6.1 Stream Classes

- **Stream Classes :**

- **1. Character Stream Classes :**

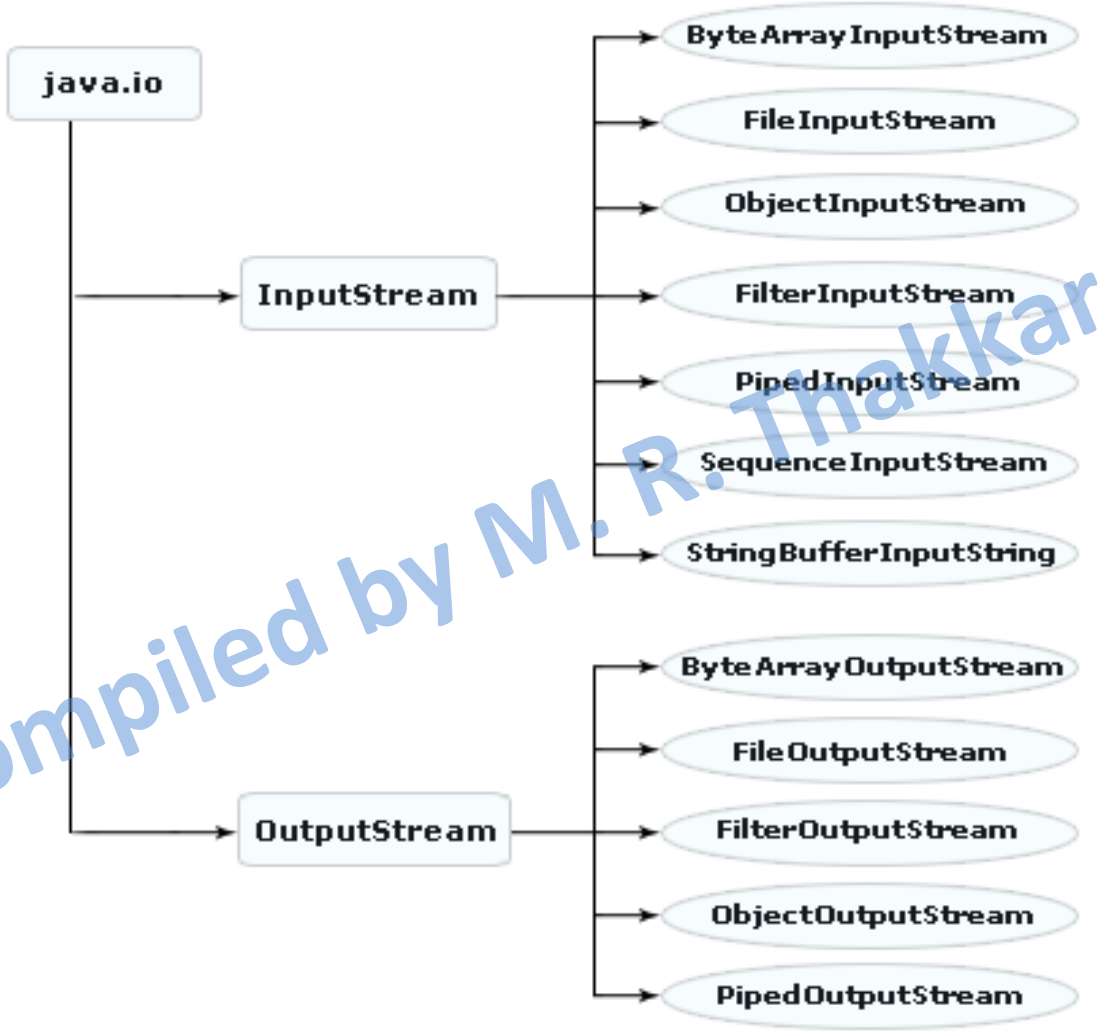
  Reader and Writer are the fundamental classes of the type character streams.

- **2. Byte Stream Classes :**

  Inputstream and Outputstream are the fundamental classes of the type Byte streams.

	Byte Streams	Character Streams
Source Streams	InputStream	Reader
Sink Streams	OutputStream	Writer

6.2 Class Hierarchy



6.3 Useful I/O Classes

- The Java Input/output (I/O) is a part of Java.io package. The Java.io package contains a relatively large number of classes that support input and output operations.

- The InputStream and OutputStream are central classes in the package which are used for reading from and writing to byte streams, respectively.

6.3 Useful I/O Classes

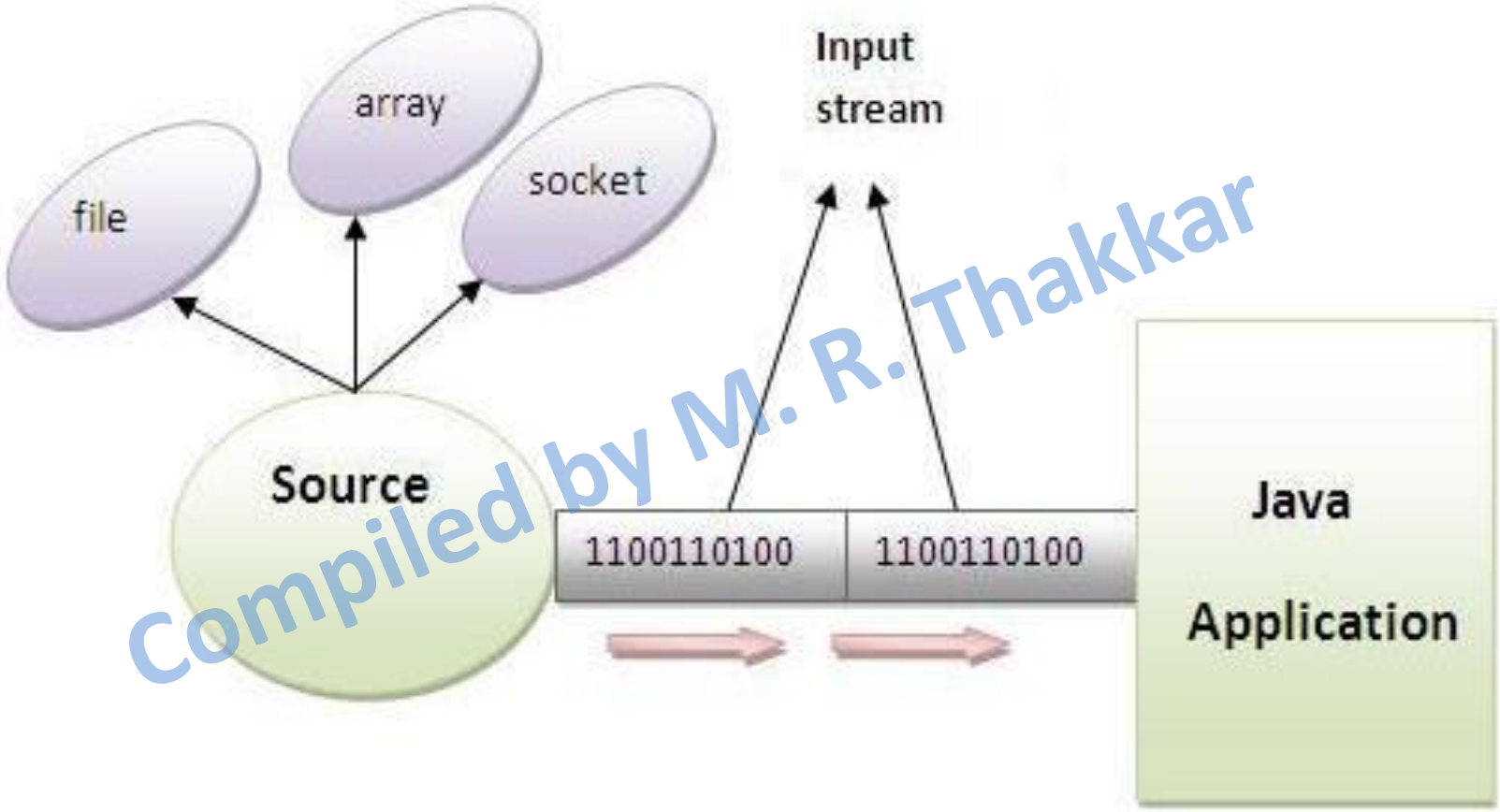
- **InputStream Class**

- Java application uses an input stream to read data from a source, it may be a file,an array, peripheral device or socket.

- An input stream is automatically opened when you create object of InputStream class.

- You can explicitly close a stream with the close( ) method, or let it be closed implicitly when the object is found as a garbage.

6.3 Useful I/O Classes



6.3 Useful I/O Classes

- **Methods**

Syntax	Description
read( )	It reads bytes of data from a stream. If no data is available it blocks the method. When a method is blocked, the thread executing waits until the data is available. It throws an IOException if an error occurs.
available( )	This method returns the number of available bytes that can be read without blockage.
close( )	This method closes the input stream. It releases the resources associated with the stream. It throws an IOException if an error occurs.
mark( )	This method is used to mark the current position in the stream.
markSupported( )	This method returns a boolean value indicating whether the stream supports mark and reset capabilities. It returns true if the stream type supports it and otherwise false.
reset( )	This method repositions the stream to the last marked position. This method throws an IOException if an error occurs.
skip( )	This method skips n bytes of input. -n is the parameter used which specifies the number of bytes to be skipped. It throws an IOException if an error occurs.

6.3 Useful I/O Classes

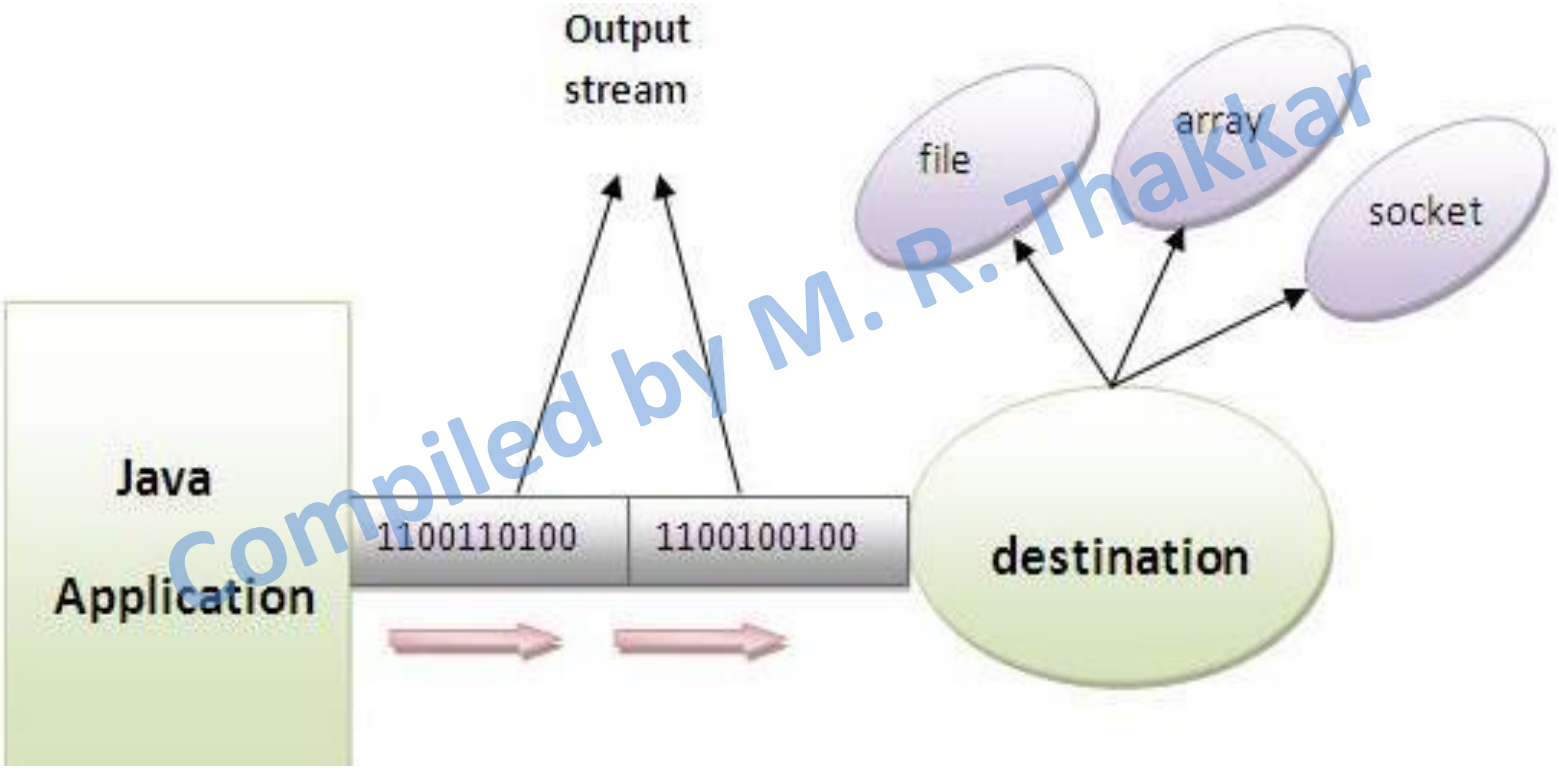
- **OutputStream Class**

- Java application uses an output stream to write data to a destination, it may be a file, an array peripheral device or socket.

- Like an input stream, an output stream is automatically opened when you create an object of OutputStream class.

- You can explicitly close an output stream with the close( ) method, or let it be closed implicitly when the object is garbage collected.

6.3 Useful I/O Classes



6.3 Useful I/O Classes

- **OutputStream Class Methods**

Syntax	Description
write( )	This method writes a byte and throws an IOException if an error occurs. This method actually blocks until the byte is written. The thread waits until the write operation is been completed.
flush( )	This method flushes the stream. The buffered data is written to the output stream. It throws an IOException if an error occurs.
close( )	This method closes the stream. It is used to release any resource associated with the stream. It throws an IOException if an error occurs.

6.4 Creation of Text file

- The **createNewFile()** method is used to create a file in Java, and return a boolean value : true if the file is created successful; false if the file is already exists or the operation failed.

- The following example shows the usage of Java.io.File.createNewFile() method.

6.4 Creation of Text file

```
import java.io.*;
public class Example
 {
      public static void main(String[] args)
      {
            try
            {
                  File file = new File("F:/newfile.txt");
                  if (!file.exists())
                  {
                        file.createNewFile();
                        System.out.println("File created Successfully.");
                  }
            }
            catch (IOException e)
            {
                  System.out.println("I/O Exception occurred.");
            }
      }
}
```

Output:

File created Successfully.

6.5 Writing Text file

- FileInputStream and FileOutputStream classes are used to read and write data in file. In another words, they are used for file handling in Java.

- **FileOutputStream class**

- A FileOutputStream use an output stream for writing data to a file.

- If you have to write primitive values then use FileOutputStream.

- Instead, for character-oriented data, prefer FileWriter.

6.5 Writing Text file

```
import java.io.*;
public class Example
{
        public static void main(String[] args)
        {
                try
                {
                        FileOutputStream fout=new FileOutputStream("F:\\newfile.txt");

                        String s="India is Great";
                         byte b[]=s.getBytes();

                        fout.write(b);
                         fout.close();

                        System.out.println("Writing Complete!");
                }
                catch(IOException e)
                {
                    System.out.println(e);
                }
        }
}
```


Output:

Writing Complete!

6.6 Reading Text file

- **FileInputStream class**

- A FileInputStream use an input stream for reading data from a file.

- If you have to read primitive values from file then use FileInputStream.

- Instead, for character-oriented data, prefer FileReader.

6.6 Reading Text file

```
import java.io.*;
public class Example
{
        public static void main(String[] args)
        {
                try
                {
                        FileInputStream fin = new FileInputStream("F:\\newfile.txt");
                        int i;

                        while((i=fin.read())!=-1)
                        {
                            System.out.print((char)i);
                        }

                        fin.close();
                }
                catch(IOException e)
                {
                    System.out.println(e);
                }
        }
}
```


Output:

India is Great

****