JAVA I/O

Prof. S. C. Sagare.

STREAMS

- Java uses the concept of **stream** to make I/O operation **fast**.
- We can perform **file handling in java** by java IO API.
- The **java.io** package contains all the classes required for input and output operations.

STREAMS

- Stream carries data from one place to another.
- Streams are categorized as
  - Input streams (InputStream class)
  - Output streams (OutputStream class)
- Input streams are streams which receive/read data.
- Output streams are streams which send/write data.

STREAMS

- There are three fields in **System** class which represent input/output device (keyboard/Monitor)
  - System.in
    - Represents **InputStream** object.
    - This object represents **standard input device**(by default **Keyboard**)
  - System.out
    - Represents **PrintStream** object.
    - This object represents **Standard output device** (by default **Monitor**)
  - System.err
    - Represents **PrintStream** object.
    - This object represents **Standard output device** (by default **Monitor**)

- Difference Between **System.out** and **System.err** ?
  - Both used to display messages on the monitor.
  - **System.out** used to display normal messages where as, **System.err** is used to display **error messages** on the monitor.

STREAMS

- Another Classification of streams is
  - **byte streams**
    - Represent data in the form of individual bytes.
  - **text streams**
    - Represent data as a character each of 2 bytes.
- If a class ends with name ‘Stream’, it comes under **byte streams**.
- If a class ends with name ‘Reader’ or ‘Writer ’, it comes under **text streams** for reading/writing text data.

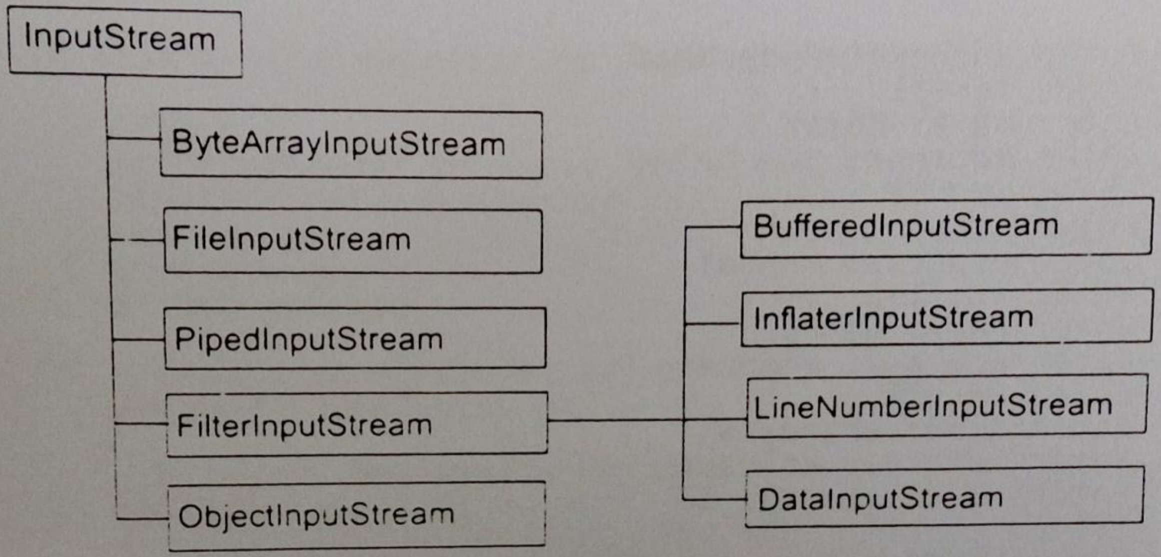

Figure 24.2(a) byte stream classes for reading data

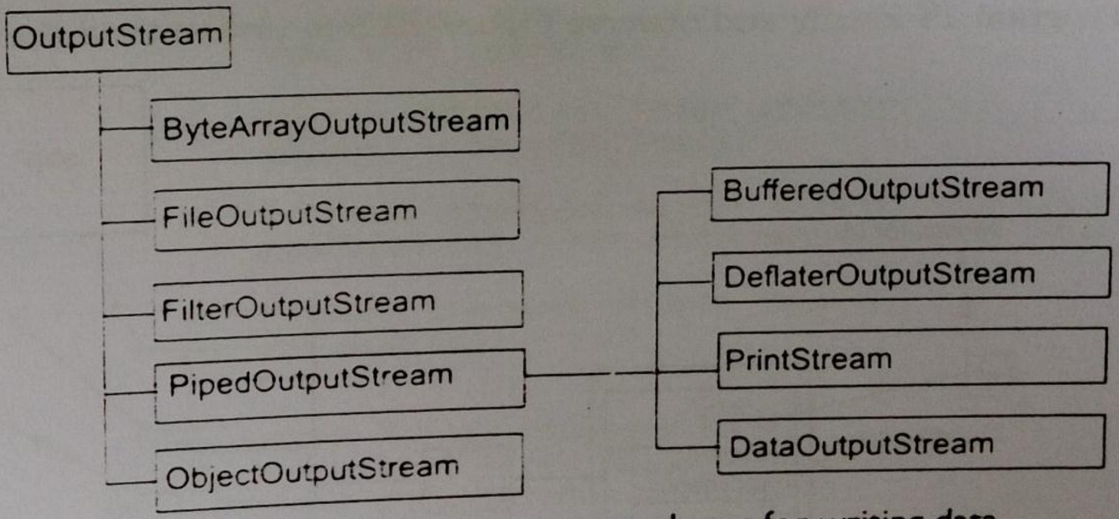

Figure 24.2(b) byte stream classes for writing data

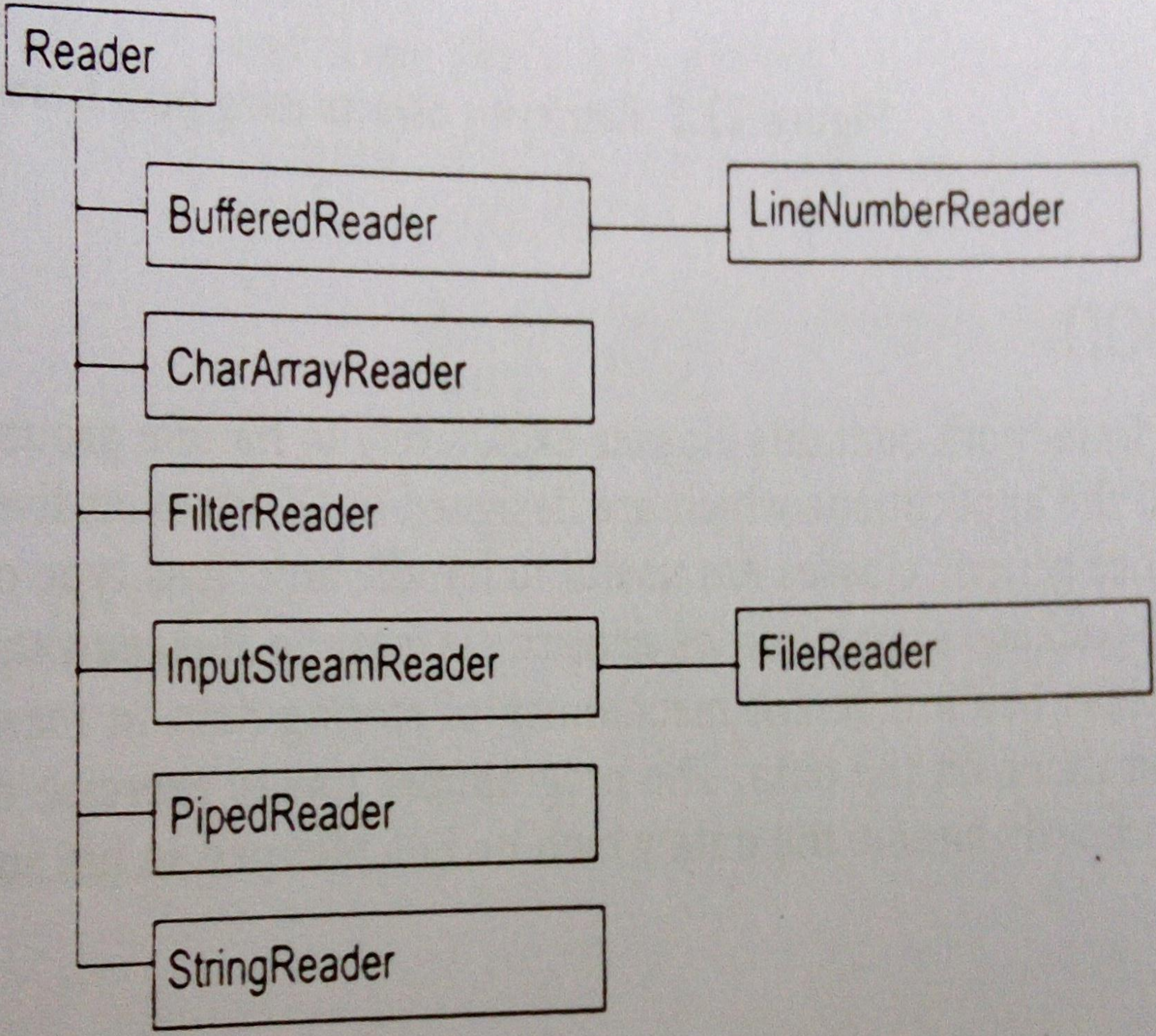

Figure 24.3(a) text stream classes for reading data

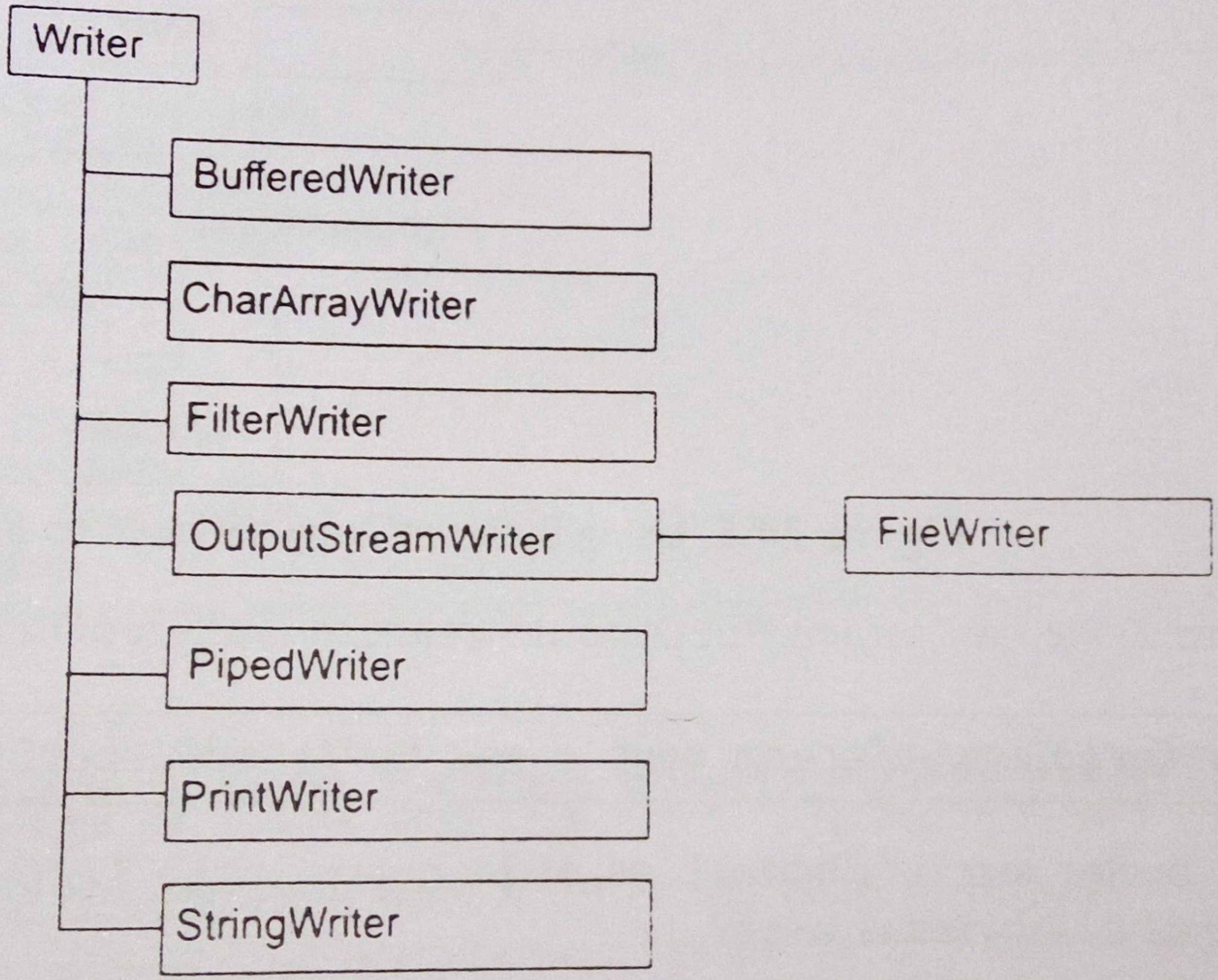

Figure 24.3(b) text stream classes for writing data

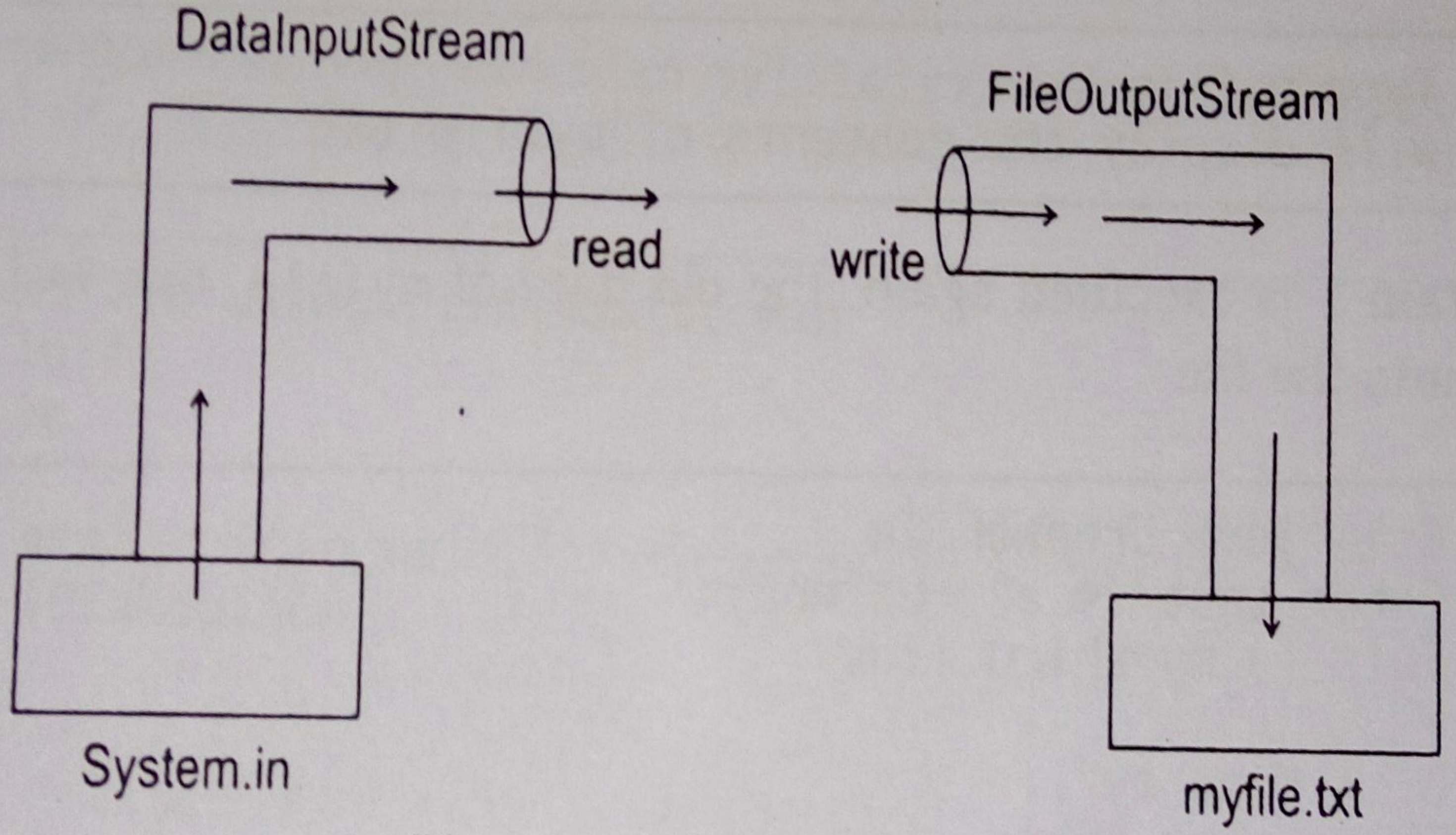

Figure 24.4 Creating a text file

➢ Java FileWriter and FileReader (File Handling in java)

- Java FileWriter and FileReader classes are used to write and read data from text files.
- These are **character-oriented** classes, used for file handling in java.
- Java has suggested not to use the FileInputStream and FileOutputStream classes if you have to read and write the textual information.

➢ Java **FileWriter** class

- Java FileWriter class is used to write character-oriented data to the file.
- Constructors of FileWriter Class

Constructor	Description
FileWriter(String file)	creates a new file. It gets file name in string.
FileWriter(File file)	creates a new file. It gets file name in File object.

Methods of **FileWriter** class

Method	Description
1) public void write(String text)	writes the string into FileWriter.
2) public void write(char c)	writes the char into FileWriter.
3) public void write(char[] c)	writes char array into FileWriter.
4) public void flush()	flushes the data of FileWriter.
5) public void close()	closes FileWriter.

```
import java.io.*;
class Simple{
 public static void main(String args[]){
  try{
   FileWriter fw=new FileWriter("abc.txt");
   fw.write("my name is sachin");
   fw.close();
  }catch(Exception e){System.out.println(e);}
  System.out.println("success");
 }
}
```

➤ Java **FileReader** class

- Java FileReader class is used to read data from the file. It returns data in byte format like FileInputStream class.

Constructor	Description
FileReader(String file)	It gets filename in string. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.
FileReader(File file)	It gets filename in file instance. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.

Methods of FileReader class

Method	Description
1) public int read()	returns a character in ASCII form. It returns -1 at the end of file.
2) public void close()	closes FileReader.

```
import java.io.*;
class Simple{
 public static void main(String args[])throws Exception{
  FileReader fr=new FileReader("abc.txt");
  int i;
  while((i=fr.read())!=-1)
  System.out.println((char)i);

  fr.close();
 }
}
```

FileInputStream and FileOutputStream (File Handling)

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

1. Java **FileOutputStream** class

- Java FileOutputStream is 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.
- But you can write byte-oriented as well as character-oriented data.

Example of Java FileOutputStream class

```
import java.io.*;
class Test{
  public static void main(String args[]){
   try{
     FileOutputstream fout=new FileOutputStream("abc.txt");
     String s="Sachin Tendulkar is my favourite player";
     byte b[]=s.getBytes();//converting string into byte array
     fout.write(b);
     fout.close();
     System.out.println("success...");
    }catch(Exception e){system.out.println(e);}
  }
}
```

2. Java **FileInputStream** class

- Java FileInputStream class obtains input **bytes from a file.**
- It is used for reading **streams of raw bytes** such as image data.
  - For reading streams of characters, consider using FileReader.
- It should be used to read byte-oriented data for example to read **image, audio, video** etc.

Example of FileInputStream class

```
import java.io.*;
class SimpleRead{
 public static void main(String args[]){
  try{
    FileInputStream fin=new FileInputStream("abc.txt");
    int i=0;
         while((i=fin.read())!=-1){
          System.out.println((char)i);
         }
    fin.close();
  }catch(Exception e){system.out.println(e);}
 }
}
```

Example of Reading the data of current java file and writing it into another file

- we can read the data of any file using the **FileInputStream** class whether it is java file, image file, video file etc.

- In next example,
    - we are reading the data of C.java file and writing it into another file M.java.

```
import java.io.*;
class C{
      public static void main(String args[])throws Exception{
            FileInputStream fin=new FileInputStream("C.java");
            FileOutputStream fout=new FileOutputStream("M.java");

          int i=0;
                  while((i=fin.read())!=-1){
                  fout.write((byte)i);
                  }
            fin.close();
      }
}
```

❖Java BufferedOutputStream and BufferedInputStream

➢ Java **BufferedOutputStream** class

- Java BufferedOutputStream class uses an internal buffer to store data.
- It adds more efficiency than to write data directly into a stream.
- So, it makes the performance fast.

Example of BufferedOutputStream class:

```
import java.io.*;
class Test{
 public static void main(String args[])throws Exception{
   FileOutputStream fout=new FileOutputStream("f1.txt");
   BufferedOutputStream bout=new BufferedOutputStream(fout)
     ;
   String s="Sachin is my favourite player";
   byte b[]=s.getBytes();
   bout.write(b);    bout.flush();    bout.close();    fout.close();
   System.out.println("success");
 }
}
```

Java BufferedInputStream class

- Java BufferedInputStream class is used to read information from stream. It internally uses buffer mechanism to make the performance fast.

```
import java.io.*;
class SimpleRead{
 public static void main(String args[]){
  try{
    FileInputStream fin=new FileInputStream("f1.txt");
    BufferedInputStream bin=new BufferedInputStream(fin);
    int i;
    while((i=bin.read())!=-1){
     System.out.println((char)i);
    }
    bin.close();
    fin.close();
  }catch(Exception e){system.out.println(e);}
 }
}
```