



Marwadi
University

Unit – 7

Input and Output

Prepared By

Prof. Ravikumar Natarajan

Assistant Professor, CE Dept.

**KNOWLEDGE IS THE CURRENCY
FOR THE 21st CENTURY**

Contents



- File
- Stream Classes
- Byte Stream Classes
- Character Stream Classes

File Handling



- A file is a **sequence of records stored in binary format**. A disk drive is formatted into several blocks that can store records. File records are mapped onto those disk blocks.
- A file is an object on a computer that stores data, information, settings, or commands used with a computer program.
- To obtain properties of file/directory.
- To delete file/directory.
- To rename file/directory.
- To create directory.
- To read File.
- To write File.
- **Absolute vs. Relative File Name**

Absolute vs Relative File Name



- An absolute file name (or full name) contains a file name with its **complete path**.
- Absolute file names are machine dependent.
- For example,
 - **Windows: D:\MEFGI\Oop1.java**
 - **directory path: D:\MEFGI, file name: Oop1.java**
 - **UNIX: /home/MEFGI/Oop1.java**
 - **directory path: /home/MEFGI, file name: Oop1.java**
- A relative file name is in relation to the **current working directory**.
- The complete directory path for a relative file name is omitted.
- For example, **Oop1.java** is a relative file name.

File Handling



- Use the **File class** to obtain file/directory properties, to delete and rename files/directories, and to create directories.
- Use the **Scanner class** for reading text data from a file.
- Use the **PrintWriter class** for writing text data to a file.

File Class



- File class is in **java.io package**.
- The File class is intended to provide an **abstraction** that deals with most of the machine-dependent complexities of files and path names in a machine-independent fashion.
- The File class contains the methods for obtaining the properties of a file/directory and for renaming and deleting a file/directory.
- **However, the File class does not contain the methods for reading and writing file contents.**

The File Class



- › File Constructors
- › exists() method
- › canRead() method
- › isDirectory() method
- › isFile() method
- › isAbsolute() method
- › isHidden() method
- › getAbsolutePath() method
- › getName() method
- › getPath() method
- › getParent() method
- › lastModified() method
- › length() method
- › listFile() method
- › delete() method
- › renameTo() method
- › mkdir() method

The Scanner Class



- › `Scanner(source: File)` // Creates a Scanner that scans tokens from the specified file.
- › `Scanner(source: String)` // Creates a Scanner that scans tokens from the specified string.
- › `close()` // Closes this scanner.
- › `hasNext(): boolean` // Returns true if this scanner has more data to be read.
- › `next(): String` // Returns next token as a string from this scanner.
- › `nextLine(): String` // Returns a line ending with the line separator from this scanner.

The Scanner Class



- › `nextByte(): byte` // Returns next token as a byte from this scanner.
- › `nextShort(): short` // Returns next token as a short from this scanner.
- › `nextInt(): int` // Returns next token as an int from this scanner.
- › `nextLong(): long` // Returns next token as a long from this scanner.
- › `nextFloat(): float` // Returns next token as a float from this scanner.
- › `nextDouble(): double` // Returns next token as a double from this scanner

Byte Streams and Character Streams



- Stream is a channel in which data flow from sender to receiver.
- Sequence of objects and methods pipelined together to produce results.
- An input object reads the stream of data from a file is called input stream.
- The output object writes the stream of data to a file is called output stream.
- These classes are found in **java.IO** package.

Byte Stream



Marwadi
University

Byte Stream

8 bits carrier

InputStream

- **BufferedInputStream**
Used for Buffered Input Stream
- **ByteArrayInputStream**
Used for reading from a byte array
- **DataInputStream**
Used for reading java standard data type
- **ObjectInputStream** - Input stream for objects
- **FileInputStream** - Used for reading from a File
- **PipedInputStream** - Input pipe
- **InputStream** - Describe stream input
- **FilterInputStream** - Implements **InputStream**

OutputStream

- **BufferedOutputStream**
Used for Buffered Output Stream
- **ByteArrayOutputStream**
Used for writing into a byte array
- **DataOutputStream**
Used for writing java standard data type
- **ObjectOutputStream** - Output stream for objects
- **FileOutputStream** - Used for writing into a File
- **PipedOutputStream** - Output pipe
- **OutputStream** - Describe stream output
- **FilterOutputStream** - Implements **OutputStream**
- **PrintStream** - Contains **print()** and **println()**

read() and **write()** both are key methods of byte stream

Character Stream



Marwadi
University

Character Stream

16 bits carrier - Unicode

Reader

- **BufferedReader**
Used for Buffered Input Stream
- **CharArrayReader**
Used for reading from an array
- **StringReader**
Used for read from a string
- **FileReader** - Used for reading from a File
- **PipedReader** - Input pipe
- **InputStreamReader** - translates bytes to character
- **FilterReader** - filtered reader
- **LineNumberReader** - used to count lines

Writer

- **BufferedWriter**
Used for Buffered Output Stream
- **CharArrayWriter**
Used for writing into an array
- **StringWriter**
Used for write into a string
- **FileWriter** - Used for writing into a File
- **PipedWriter** - Output pipe
- **OutputStreamWriter** - characters to bytes
- **FilterWriter** - filtered writer
- **PrintStream** - Contains **print()** and **println()**

read() and **write()** both are key methods of byte stream

ByteStream VS CharacterStream



Character streams

1. Meant for reading or writing to character- or text-based I/O such as text files, text documents, XML, and HTML files.
2. Data dealt with is 16-bit Unicode characters.
3. Input and output character streams are called readers and writers, respectively.
4. The abstract classes of Reader and Writer and their derived classes in the java.io package provide support for character streams.

Byte streams

1. Meant for reading or writing to binary data I/O such as executable files, image files, and files in low-level file formats such as .zip, .class, .obj and .exe.
2. Data dealt with is bytes (i.e., units of 8-bit data).
3. Input and output byte streams are simply called input streams and output streams, respectively.
4. The abstract classes of Input Stream and Output Stream and their derived classes in the java.io package provide support for byte streams.

Read and write operations on file using InputStream and OutputStream



//To write into a file using byte stream

```
import java.io.*;
class output
{public static void main(String args[])
{      String s="This is my file";
int a=5;
Double d=5.35;
try
{
FileOutputStream fos= new FileOutputStream("abcd.txt");
DataOutputStream dos = new DataOutputStream(fos);
dos.writeBytes(s);
dos.writeInt(a);
dos.writeDouble(d);
dos.close();
}
catch(IOException ex)
{ex.printStackTrace();}
}}
```

Read and write operations on file using InputStream and OutputStream



//To read from a file using byte stream

```
import java.io.*;
class input
{public static void main(String args[])
{
try
{
    FileInputStream fin= new FileInputStream("abcd.txt"); //to read data from a file in bytes
    DataInputStream din = new DataInputStream(fin); // read primitive Java data types
        String line=null;
        while((line =din.readLine())!=null)
        {
            System.out.println(line);
        }
        din.close();
    }
catch(Exception ex)
{ex.printStackTrace();}
}}
```

FileWriter and FileReader



//To write into a file using character stream

```
import java.io.*;
class writerDemo
{
    public static void main(String[] args)
    {
        try
        {
            FileWriter fw = new FileWriter("abc.txt");
            fw.write("Hello, Good Morning"); // fw.write("123");
            fw.close();
        }
        catch(IOException ex)
        {ex.printStackTrace();}
    }
}
```

//To read from a file using character stream

```
import java.io.*;
class readerDemo
{
    public static void main(String[] args)
    {
        try{
            File f1= new File("abc.txt");
            FileReader fr = new FileReader(f1);

            BufferedReader br=new BufferedReader(fr);
            //chaining
            String line=null;
            while((line=br.readLine()) !=null)
            {
                System.out.println(line);
            }
            br.close();
        }
        catch(Exception ex)
        {ex.printStackTrace();} }
}
```


Questions



- File?
- What is a Stream?
- Two types of Streams?
- Byte Stream Classes used for?
- Character Stream Classes used for?

Summary



- File
- Stream Classes
- Byte Stream Classes
- Character Stream Classes

Next



Marwadi
University

- String class
- Character class
- StringBuffer class
- StringBuilder class
- Primitive type Wrapper classes
- Collections overview
- Collection interfaces
- Collection classes
- Maps
- Comparators
- Lists
- Vector class
- Stack class
- Scanner
- Formatter



Marwadi
University

END OF UNIT - 7

KNOWLEDGE IS THE CURRENCY
FOR THE 21st CENTURY

REFERENCE PURPOSE



Marwadi
University

REFERENCE PURPOSE

Example 2

PrintWriter and Reader



```
import java.io.PrintWriter;
```

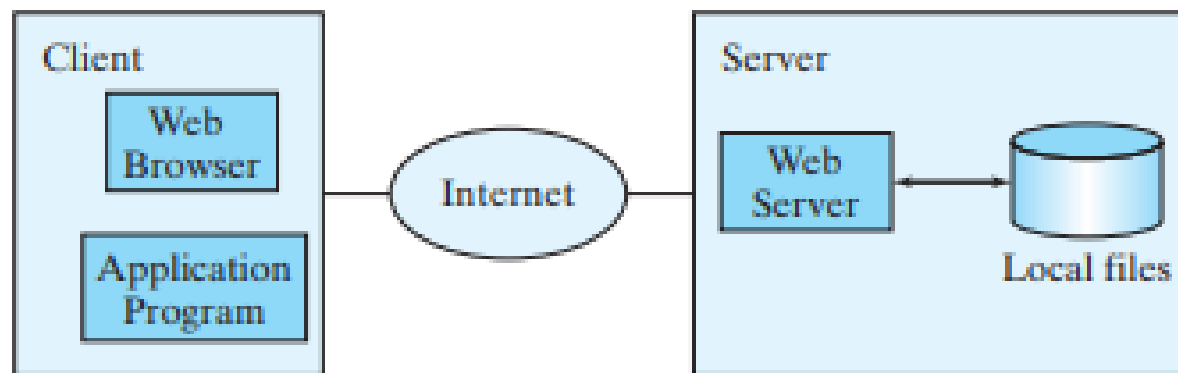
```
class Main {  
    public static void main(String[] args) {  
  
        String data = "This is a text inside the file.";  
  
        try {  
            PrintWriter output = new PrintWriter("PQR.txt");  
  
            output.print(data);  
            output.close();  
        }  
        catch(Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
import java.io.*;  
public class ReaderExample {  
    public static void main(String[] args) {  
        try {  
            Reader reader = new FileReader("PQR.txt");  
            int data = reader.read();  
            while (data != -1) {  
                System.out.print((char) data);  
                data = reader.read();  
            }  
            reader.close();  
        } catch (Exception ex) {  
            System.out.println(ex.getMessage());  
        }  
    }  
}
```

Reading Data from the Web



- Just like you can read data from a file on your computer, you can read data from a file on the Web.
- You can also access data from a file that is on the Web if you know the file's **URL (Uniform Resource Locator)**—the unique address for a file on the Web).
- For example, **www.google.com/index.html** is the URL for the file index.html located on the Google Web server.



The client retrieves files from a Web server.

Reading Data from the Web



- For an application program to read data from a URL, you first need to create a URL object using the `java.net.URL` class with this constructor:
- `public URL(String spec) throws MalformedURLException`

```
try {  
    URL url = new URL("http://www.google.com/index.html");  
}  
catch (MalformedURLException ex)  
{ ex.printStackTrace();  
}
```


Reading Data from the Web



- For an application program to read data from a URL, you first need to create a URL object using the `java.net.URL` class with this constructor:
- `public URL(String spec) throws MalformedURLException`

```
try {  
    URL url = new URL("http://www.google.com/index.html");  
}  
catch (MalformedURLException ex)  
{ ex.printStackTrace();  
}
```

Reading Data from the Web



```
import java.net.*;  
import java.io.*;
```

```
public class ReadURL {  
    public static void main(String[] args) throws Exception {
```

```
        URL url = new URL("http://www.google.com/index.html");  
        BufferedReader read = new BufferedReader(  
            new InputStreamReader(url.openStream()));
```

```
        String i;  
        while ((i = read.readLine()) != null)  
            System.out.println(i);  
        read.close();  
    }  
}
```

Abstract Class VS Interface



Refer previous unit for examples.

Parameters	Interface	Abstract class
Speed	Slow	Fast
Multiple Inheritances	Implement several Interfaces	Only one abstract class
Structure	Abstract methods	Abstract & concrete methods
When to use	Future enhancement	To avoid independence
Inheritance/ Implementation	A Class can implement multiple interfaces	The class can inherit only one Abstract Class
Data fields	the interface cannot contain data fields.	the class can have data fields.
Abstract keyword	In an abstract interface keyword, is optional for declaring a method as an abstract.	In an abstract class, the abstract keyword is compulsory for declaring a method as an abstract.

Abstract Class Example

Ex.: Shape (superclass), Circle and Rectangle (subclass)

Shape Class

```
public abstract class Shape {  
    ...  
    ...  
  
    /** Abstract method getArea */  
    public abstract double getArea();  
  
    /** Abstract method getPerimeter */  
    public abstract double getPerimeter();  
  
}
```

Main Class

Class main{

Circle c = new circle();

Rectangle r = new Rectangle(); }



Marwadi
University

Circle Class

```
public class Circle extends Shape {  
    ...  
    ...  
  
    public double getArea(){  
        //Implementation  
    }  
    public double getPerimeter(){  
        //Implementation  
    }  
}
```

Rectangle Class

```
public class Rectangle extends Shape {  
    ...  
    ...  
  
    public double getArea(){  
        //Implementation  
    }  
    public double getPerimeter(){  
        //Implementation  
    }  
}
```

Interface



- Since abstract class allows concrete methods as well, it does not provide 100% abstraction.
- You can say that it provides partial abstraction.
- Interfaces are used for 100% abstraction (full abstraction)

Syntax:

```
modifier interface InterfaceName {
```

```
/** Constant declarations */
```

```
/** Abstract method signatures */
```

```
}
```

The Comparable Interface



- Suppose you want to design a generic method to find the larger of two objects of the same type, such as two students / dates / circles / rectangles / etc.
- In order to accomplish this, the two objects must be comparable, so the common behavior for the objects must be comparable.
- Java provides the Comparable interface for this purpose.
- The Comparable interface defines the **compareTo** method for comparing objects.

The interface is defined as follows:

```
package java.lang;  
public interface Comparable<E> {  
    public int compareTo(E o);  
}
```

The Comparable Interface



- The Comparable interface is a **generic** interface.
- The generic type E [Comparable<E>] is replaced by a concrete type when implementing this interface.

```
class circle implements Comparable<Circle> {  
    public int compareTo(Circle o){  
        ...  
    }  
}
```

The Comparable Interface

SortComparableObjects



```
import java.math.*;
public class Main {
    public static void main(String[] args) {
        String[] cities = {"Savannah", "Boston", "Atlanta", "Tampa"};
        java.util.Arrays.sort(cities);
        for (String city: cities)
            System.out.print(city + " ");
        System.out.println();
    }
}
```

Output:

Atlanta Boston Savannah Tampa

54623239292 432232323239292 2323231092923992

```
BigInteger[] hugeNumbers = {new BigInteger("2323231092923992"),
    new BigInteger("432232323239292"),
    new BigInteger("54623239292")};
java.util.Arrays.sort(hugeNumbers);
for (BigInteger number: hugeNumbers)
    System.out.print(number + " ");
}}
```


The Comparable Interface



```
class Student implements Comparable<Student>{  
    int rollno;  
    String name;  
    int age;  
    Student(int rollno,String name,int age){  
        this.rollno=rollno;  
        this.name=name;  
        this.age=age;  
    }
```

```
    public int compareTo(Student st){  
        if(age==st.age)  
            return 0;  
        else if(age>st.age)  
            return 1;  
        else  
            return -1;  
    }  
}
```

```
import java.util.*;  
public class TestSort2{  
    public static void main(String args[]){  
        ArrayList<Student> al=new ArrayList<Student>  
        ();  
        al.add(new Student(101,"Vijay",23));  
        al.add(new Student(106,"Ajay",27));  
        al.add(new Student(105,"Jai",21));  
  
        Collections.sort(al);  
        for(Student st:al){  
            System.out.println(st.rollno+" "+st.name+" "+st.  
            age);  
        }  
    }  
}
```

The Cloneable Interface



- Often it is desirable to create a copy of an object. To do this, you need to use the clone method and understand the Cloneable interface.
- **The Cloneable interface specifies that an object can be cloned.**
- An interface contains constants and abstract methods, but the Cloneable interface is a special case. The Cloneable interface in the **java.lang** package is defined as follows:

```
package java.lang;  
public interface Cloneable {  
}
```

This interface is empty. An interface with an empty body is referred to as a marker interface. A marker interface does not contain constants or methods. It is used to denote that a class possesses certain desirable properties. A class that implements the Cloneable interface is marked cloneable, and its objects can be cloned using the clone() method defined in the Object class.

The Cloneable Interface



```
package java.lang;  
public interface Cloneable {  
}
```

- This interface is empty. An interface with an empty body is referred to as a **marker interface**.
- A marker interface does not contain constants or methods. It is used to denote that a class possesses certain desirable properties.
- A class that implements the Cloneable interface is marked cloneable, and its objects can be cloned using the clone() method defined in the Object class.
- **Many classes in the Java library (e.g., Date, Calendar, ArrayList etc.)** implement Cloneable. Thus, the instances of these classes can be cloned.

The Cloneable Interface



- **Creating Copy of Java Object**
- We can create a replica or copy of java object by
- Creating a copy of object in a different memory location. This is called a **Deep copy**.
- Creating a new reference that points to the same memory location. This is also called a **Shallow copy**.

The Cloneable Interface



```
//ShallowCopy
class Main
{
    int x = 30;

    public static void main(String args[])
    {
        Main obj1 = new Main();

        // it will copy the reference, not value
        Main obj2 = obj1;

        obj2.x = 6;
        System.out.println("The value of x is: " + obj1.x);
    } }
```

```
//DeepCopy
class Main implements Cloneable
{
    public int x = 30;

    public static void main(String args[])
    {
        Main obj1 = new Main();

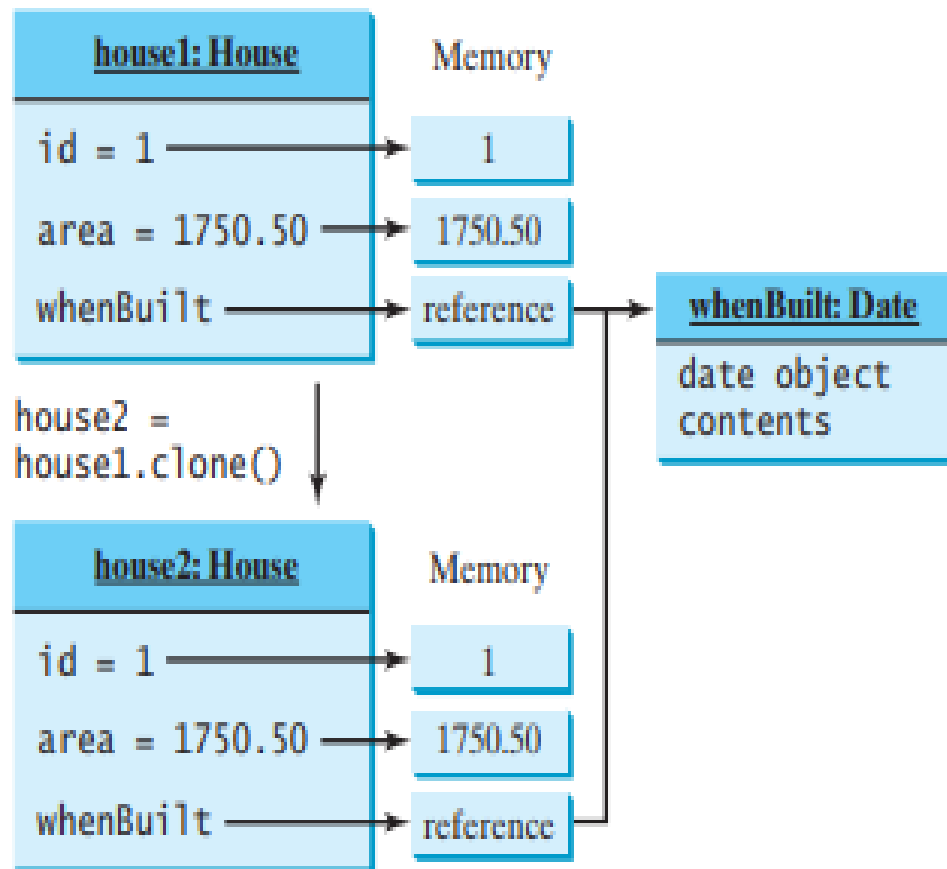
        // it will copy the reference, not value
        //SCopy obj2 = obj1;
        try{
            Main obj2 = (Main)obj1.clone();
            obj2.x = 6;
            System.out.println("The value of x is: " + obj1.x);
            System.out.println("The value of x is: " + obj2.x);
        }
        catch(Exception e){System.out.println(e);}

    } }
```

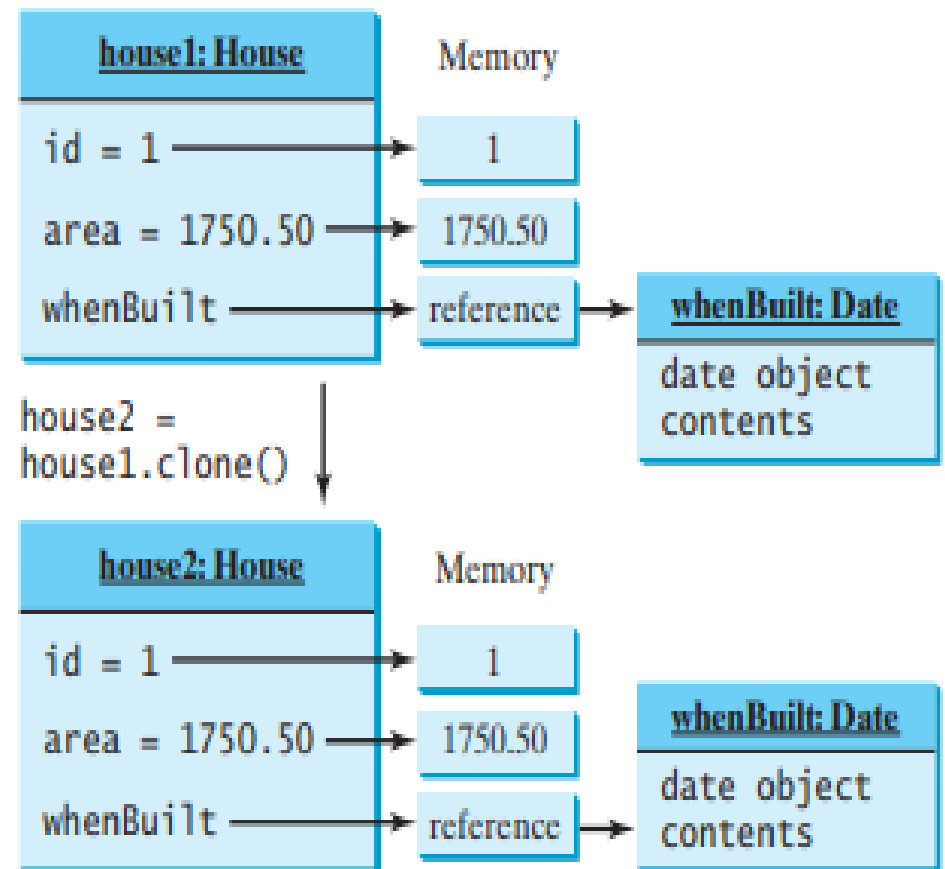
The Cloneable Interface



Marwadi
University



Shallow Copy



Deep Copy



Marwadi
University

Reference Programs

KNOWLEDGE IS THE CURRENCY
FOR THE 21st CENTURY

Reading Data from the Web



```
import java.util.Scanner;
```

```
public class ReadFileFromURL {  
    public static void main(String[] args) {  
        System.out.print("Enter a URL: ");  
        String URLString = new Scanner(System.in).next();
```

```
    try {  
        java.net.URL url = new java.net.URL(URLString);  
        int count = 0;  
        Scanner input = new Scanner(url.openStream());  
        while (input.hasNext()) {  
            String line = input.nextLine();  
            count += line.length();  
        }
```

```
        System.out.println("The file size is " + count + " characters");  
    }  
    catch (java.net.MalformedURLException ex) {  
        System.out.println("Invalid URL");  
    }  
    catch (java.io.IOException ex) {  
        System.out.println("I/O Errors: no such file");  
    }  
}
```


ByteStream VS CharacterStream



```
import java.io.*;
class CopyFile
{
    public static void main(String args[])
    {
        try
        {
            FileInputStream fr = new FileInputStream("photo.jpg");
            FileOutputStream fw = new FileOutputStream("copy.jpg");
            int i = 0;
            while ((i = fr.read()) != -1) {
                fw.write(i);
            }
            fw.flush();
            fw.close();
            fr.close();
            System.out.println("File copied successfully.....");
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
}
```

- First place the image in the proper folder.
- This program will copy the picture as it is in the name of "copy".
- This will work fine.

ByteStream VS CharacterStream



```
import java.io.*;
```

```
class CopyFile
```

```
{
```

```
    public static void main(String args[])
```

```
    {
```

```
        try
```

```
        {
```

```
            FileReader fr = new FileReader("Data.txt");
```

```
            FileWriter fw = new FileWriter("Copy.txt");
```

```
            int i = 0;
```

```
            while ((i=fr.read())!=-1){
```

```
                fw.write(i);
```

```
            }
```

```
            fw.flush();
```

```
            fw.close();
```

```
            fr.close();
```

```
            System.out.println("File copied successfully.....");
```

```
        }
```

```
        catch(Exception e)
```

```
        {
```

```
            System.out.println(e);
```

```
        }
```

```
    } }
```

- **TRY**
 - First place the image in the proper folder.
 - Try to create a copy as like previous program it will not copy the image.
 - **The copied image will be corrupted.**
- **This will work fine for the text.**

ByteStream VS CharacterStream



- Figure out the reasons for the previous two programs.



Marwadi
University

Thank you

End of Unit 7