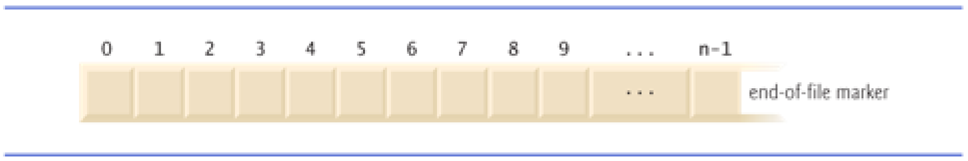
Java & JEE Training

Day 22 – Java IO – Files, Streams and Object Serialization

MindsMapped Consulting

Files and Streams

- Java views each file as a sequential **stream of bytes**



- Every operating system provides a mechanism to determine the end of a file, such as an **end-of-file marker** or a count of the total bytes in the file that is recorded in a system-maintained administrative data structure.
- A Java program simply receives an indication from the operating system when it reaches the end of the stream

Files and Streams Contd..

- File streams can be used to input and output data as bytes or characters.
- Streams that input and output bytes are known as **byte-based streams**, representing data in its binary format.
- Streams that input and output characters are known as **character-based streams**, representing data as a sequence of characters.
- Files that are created using byte-based streams are referred to as **binary files.**
- Files created using character-based streams are referred to as **text files.** Text files can be read by text editors.
- Binary files are read by programs that understand the specific content of the file and the ordering of that content.

Files and Streams (Contd.)

- A Java program **opens** a file by creating an object and associating a stream of bytes or characters with it.
  - Can also associate streams with different devices.
- Java creates three stream objects when a program begins executing
  - `System.in` (the standard input stream object) normally inputs bytes from the keyboard
  - `System.out` (the standard output stream object) normally outputs character data to the screen
  - `System.err` (the standard error stream object) normally outputs character-based error messages to the screen.

Files and Streams (Contd.)

- Java programs perform file processing by using classes from package **java.io**.
- Includes definitions for stream classes
  - **FileInputStream** (for byte-based input from a file)
  - **FileOutputStream** (for byte-based output to a file)
  - **FileReader** (for character-based input from a file)
  - **FileWriter** (for character-based output to a file)
- You open a file by creating an object of one these stream classes. The object’s constructor opens the file.

Files and Streams (Contd.)

- Can perform input and output of objects or variables of primitive data types without having to worry about the details of converting such values to byte format.
- To perform such input and output, objects of classes **ObjectInputStream** and **ObjectOutputStream** can be used together with the byte-based file stream classes `FileInputStream` and `FileOutputStream`.
- The complete hierarchy of classes in package `java.io` can be viewed in the online documentation at
- http://java.sun.com/javase/6/docs/api/java/io/package-tree.html

Class “File”

```
// Demonstrating the File class.
import java.io.File;

public class FileDemonstration
{
   // display information about file user specifies
   public void analyzePath( String path )
   {
      // create File object based on user input
      File name = new File( path );

      if ( name.exists() ) // if name exists, output information about it
      {
         // display file (or directory) information
         System.out.printf(
            "%s%s\n%s\n%s\n%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s",
            name.getName(), " exists",
            ( name.isFile() ? "is a file" : "is not a file" ),
            ( name.isDirectory() ? "is a directory" :
               "is not a directory" ),
            ( name.isAbsolute() ? "is absolute path" :
               "is not absolute path" ), "Last modified: ",
            name.lastModified(), "Length: ", name.length(),
            "Path: ", name.getPath(), "Absolute path: ",
            name.getAbsolutePath(), "Parent: ", name.getParent() );

         if ( name.isDirectory() ) // output directory listing
         {
            String directory[] = name.list();
            System.out.println( "\n\nDirectory contents:\n" );

            for ( String directoryName : directory )
               System.out.printf( "%s\n", directoryName );
         } // end else
      } // end outer if
      else // not file or directory, output error message
      {
         System.out.printf( "%s %s", path, "does not exist." );
      } // end else
   } // end method analyzePath
} // end class FileDemonstration
```

Class File

- A **separator character** is used to separate directories and files in the path.
- On Windows, the separator character is a backslash (`\`).
- On Linux/UNIX, it's a forward slash (`/`).
- Java processes both characters identically in a path name.
- When building `String`s that represent path information, use `File.separator` to obtain the local computer's proper separator.
    - This constant returns a `String` consisting of one character—the proper separator for the system.

Java IO Classes for streaming
Byte-based and Character-based

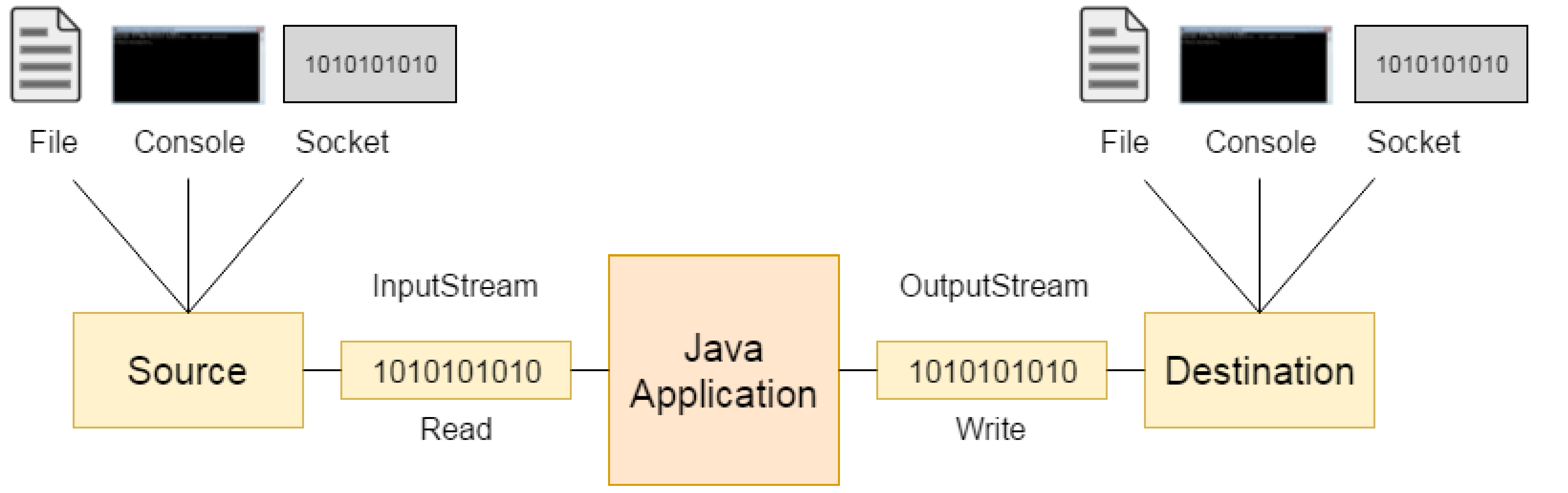
java.io classes

- Let us look at some additional interfaces and classes (from package `java.io`) for byte-based input and output streams and character-based input and output streams.

Standard streams available by default

- **System.out:** standard output stream
- **System.in:** standard input stream
- **System.err:** standard error stream

InputStream and OutputStream



Interfaces and Classes for Byte-Based Input and Output

- **InputStream** and **OutputStream** are `abstract` classes that declare methods for performing byte-based input and output, respectively.

- Concrete implementations:
  - FileInputStream and FileOutputStream
  - BufferedInputStream and BufferedOutputStream
  - ObjectInputStream and ObjectOutputStream

Interfaces and Classes for Character-Based Input and Output

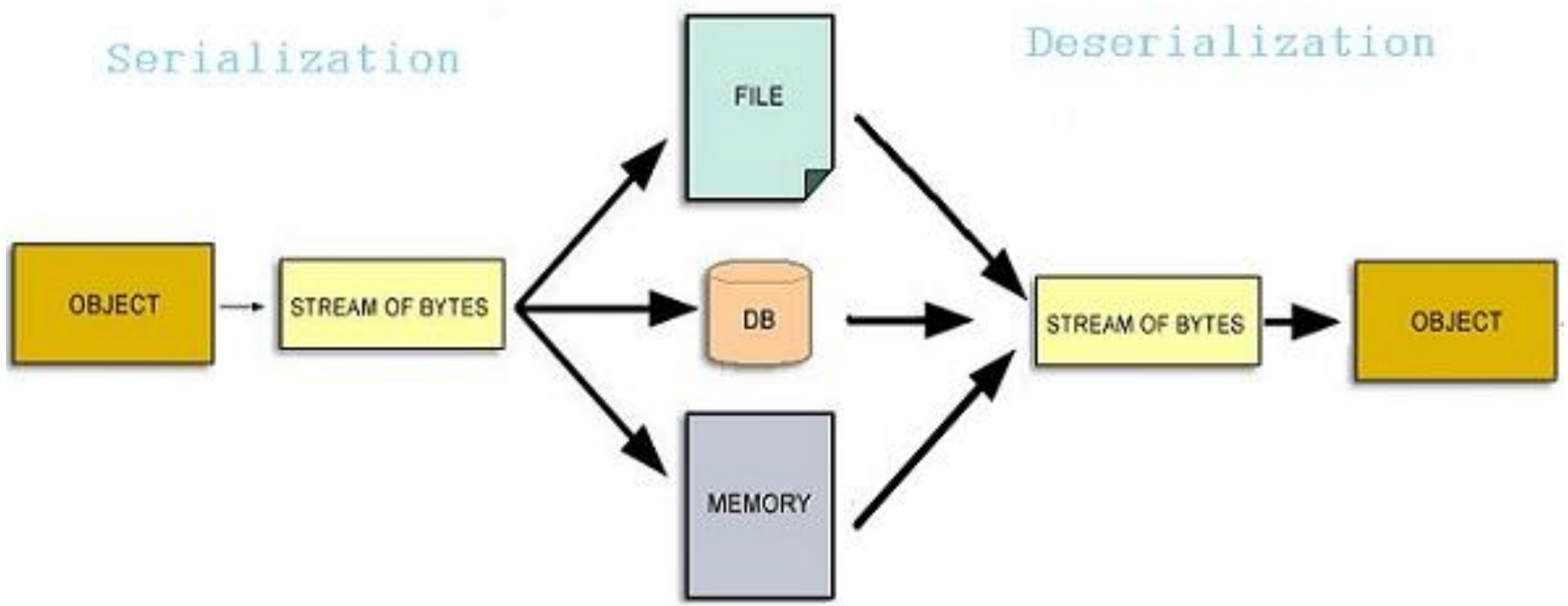
- The **Reader** and **Writer** `abstract` classes are Unicode two-byte, character-based streams.
- Most of the byte-based streams have corresponding character-based concrete `Reader` or `Writer` classes.
- Concrete Implementations:
  - FileReader and FileWriter
  - BufferedReader and BufferedWriter

Remember..

- Always prefer Buffered I/O over non-buffered I/O. It yields significant performance improvement over unbuffered I/O, unless proven otherwise.

Object Serialization

Object Serialization and Deserialization



Object Serialization

- To read an entire object from or write an entire object to a file, Java provides **object serialization.**
- A **serialized object** is represented as a sequence of bytes that includes the object’s data and its type information.
- After a serialized object has been written into a file, it can be read from the file and **deserialized** to recreate the object in memory.

Object Serialization

- In a class that implements `Serializable`, every variable must be `Serializable`.
- Any one that is not must be declared **transient** so it will be ignored during the serialization process.
- All primitive-type variables are serializable.
- For reference-type variables, check the class's documentation (and possibly its superclasses) to ensure that the type is `Serializable`.
- Serializable is a marker interface… it has no methods defined so no method needs to be overridden. It is just an instruction to the JRE that this class is Serializable.

Serialization – How to serialize objects in Java?

- ObjectOutputStream output = new ObjectOutputStream(new FileOutputStream(“myfileforstoringobject.txt”));

- output.writeObject(record); //record = new instance of the serializable object

- Caution: It’s a logic error to open an existing file for output when, in fact, you wish to preserve the file. Class FileOutputStream provides an overloaded constructor to open and append the data, instead of overwriting. This will preserve the contents of the file. Try this constructor as well.

Deserialization: How to deserialize in Java?

- ObjectInputStream input = new ObjectInputStream(new FileInputStream(“clients.txt”));
- record = (SerializableClassName) input.readObject(); //record is instance of the serializable class that we are trying to read from the file.

Best Practice to avoid resource leaks… < JDK 7

```
try{
    //use buffering
    OutputStream file = new FileOutputStream("quarks.ser");
    OutputStream buffer = new BufferedOutputStream(file);
    ObjectOutput output = new ObjectOutputStream(buffer);
    try{
      output.writeObject(quarks);
    }
    finally{
      output.close();
    }
  }
  catch(IOException ex){
    fLogger.log(Level.SEVERE, "Cannot perform output.", ex);
  }
```

Best Practice to avoid resource leaks… < JDK 7

```
//use buffering
OutputStream file = new FileOutputStream("quarks.ser");
OutputStream buffer = new BufferedOutputStream(file);
ObjectOutput output = new ObjectOutputStream(buffer);
try{
  output.writeObject(quarks);
} catch(IOException ex){
     //handle exception code
}
finally{
 try{
  output.close();
 }catch(IOException ex){
     //handle exception code
}


}
```

Best practice for closing resources … JDK 7 onwards

- The **try-with-resources statement** ensures that each resource is closed at the end of the statement, you do not have to explicitly close the resources.
- Need not explicitly call close()

```
import java.io.*;
class Test
{
    public static void main(String[] args){
        try(BufferedReader br=new BufferedReader(new FileReader("myfile.txt"))){
            String str;
            while((str=br.readLine())!=null){
                System.out.println(str);
            }
        }catch(IOException ie){
             System.out.println("exception");
        }
    }
}
```

Exercise…

- Write a class in line with AccountRecord.java, but this time make it a serializable class.
- Create 5 objects and write those into a text file.
- Read each of the 5 objects

Miscellaneous examples

FileReader and FileWriter Example

```
import java.io.*;
public class FileRead {

   public static void main(String args[])throws IOException {
      File file = new File("Hello1.txt");

      // creates the file
      file.createNewFile();

      // creates a FileWriter Object
      FileWriter writer = new FileWriter(file);

      // Writes the content to the file
      writer.write("This\n is\n an\n example\n");
      writer.flush();
      writer.close();

      // Creates a FileReader Object
      FileReader fr = new FileReader(file);
      char [] a = new char[50];
      fr.read(a);   // reads the content to the array

      for(char c : a)
         System.out.print(c);   // prints the characters one by one
      fr.close();
   }
}
```

File Example: Creating Directories

```
import java.io.File;
public class CreateDir {

  public static void main(String args[]) {
    String dirname = "/tmp/user/java/bin";
    File d = new File(dirname);

    // Create directory now.
    d.mkdirs(); //Question: What does mkdirs do?
  }
}
```

File Example: Listing Directories

```
import java.io.File;
public class ReadDir {

  public static void main(String[] args) {
    File file = null;
    String[] paths;

    try {
      // create new file object
      file = new File("/tmp");

      // array of files and directory
      paths = file.list();

      // for each name in the path array
      for(String path:paths) {
        // prints filename and directory name
        System.out.println(path);
      }
    }catch(Exception e) {
      // if any error occurs
      e.printStackTrace();
    }
  }
}
```

File Example: Create a file

```
import java.io.File;
import java.io.IOException;

public class CreateFileDemo
{
   public static void main( String[] args )
   {
      try {
                File file = new File("C:\\newfile.txt");
                /*If file gets created then the createNewFile()
                 * method would return true or if the file is
                 * already present it would return false
                 */
             boolean fvar = file.createNewFile();
                if (fvar){
                     System.out.println("File has been created successfully");
                }
                else{
                     System.out.println("File already present at the specified location");
                }
          } catch (IOException e) {
                         System.out.println("Exception Occurred:");
                    e.printStackTrace();
           }
   }
}
```

How to read file in Java – BufferedInputStream

```
import java.io.*;
public class ReadFileDemo {
   public static void main(String[] args) {
      //Specify the path of the file here
      File file = new File("C://myfile.txt");
      BufferedInputStream bis = null;
      FileInputStream  fis= null;

      try
      {
         //FileInputStream to read the file
         fis = new FileInputStream(file);

         /*Passed the FileInputStream to
BufferedInputStream
          *For Fast read using the buffer array.*/
         bis = new BufferedInputStream(fis);

         /*available() method of BufferedInputStream
          * returns 0 when there are no more bytes
          * present in the file to be read*/
         while( bis.available() > 0 ){
            System.out.print((char)bis.read());
         }

       }
catch(FileNotFoundException fnfe)
        {
           System.out.println("The specified file not found" +
fnfe);
        }
        catch(IOException ioe)
        {
           System.out.println("I/O Exception: " + ioe);
        }
        finally
        {
           try{
              if(bis != null && fis!=null)
              {
                           fis.close();
                 bis.close();
              }
            }catch(IOException ioe)
             {
                 System.out.println("Error in InputStream close():
" + ioe);
             }
        }
     }
}
```

How to read file in Java using BufferedReader

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadFileDemo {
  public static void main(String[] args) {

      BufferedReader br = null;
      BufferedReader br2 = null;
      try{
         br = new BufferedReader(new FileReader("B:\\myfile.txt"));


         //One way of reading the file
                        System.out.println("Reading the file using readLine()
method:");
                        String contentLine = br.readLine();
                        while (contentLine != null) {
                           System.out.println(contentLine);
                           contentLine = br.readLine();
                        }

                        br2 = new BufferedReader(new
FileReader("B:\\myfile2.txt"));

                        //Second way of reading the file
                        System.out.println("Reading the file using read()
method:");
 int num=0;
                        char ch;
                        while((num=br2.read()) != -1)
                        {
              ch=(char)num;
                           System.out.print(ch);
                        }

      }
catch (IOException ioe)
      {
                        ioe.printStackTrace();
      }
      finally
      {
                        try {
                          if (br != null)
                                          br.close();
                          if (br2 != null)
                                          br2.close();
                        }
                        catch (IOException ioe)
          {
                                          System.out.println("Error in
closing the BufferedReader");
                        }
                      }
   }
}
```

How to write to file in Java using BufferedWriter

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class WriteFileDemo {
   public static void main(String[] args) {
      BufferedWriter bw = null;
      try {
                   String mycontent = "This String would be
written" +
                      " to the specified File";
         //Specify the file name and path here
                   File file = new File("C:/myfile.txt");

                   /* This logic will make sure that the file
                    * gets created if it is not present at the
                    * specified location*/
                    if (!file.exists()) {
                       file.createNewFile();
                    }

                   FileWriter fw = new FileWriter(file);
                   bw = new BufferedWriter(fw);
                   bw.write(mycontent);
        System.out.println("File written Successfully");

    } catch (IOException ioe) {
                   ioe.printStackTrace();
                }
                finally
                {
                   try{
                      if(bw!=null)
                                   bw.close();
                   }catch(Exception ex){
                       System.out.println("Error in closing the
BufferedWriter"+ex);
                    }
                }
   }
}
```

Append content to File using FileWriter and BufferedWriter

```
import java.io.File;
import java.io.FileWriter;
import java.io.BufferedWriter;
import java.io.IOException;

class AppendFileDemo
{
   public static void main( String[] args )
   {
      try{
                   String content = "This is my content which
would be appended " +
                   "at the end of the specified file";
        //Specify the file name and path here
                   File file =new File("C://myfile.txt");

                   /* This logic is to create the file if the
                    * file is not already present
                    */
                   if(!file.exists()){
                      file.createNewFile();
                   }

                   //Here true is to append the content to file
                   FileWriter fw = new FileWriter(file,true);
                   //BufferedWriter writer give better
performance
                   BufferedWriter bw = new
BufferedWriter(fw);
                   bw.write(content);
                   //Closing BufferedWriter Stream
                   bw.close();

                   System.out.println("Data successfully
appended at the end of file");

      }catch(IOException ioe){
         System.out.println("Exception occurred:");
                    ioe.printStackTrace();
       }
   }
}
```

Append content to File using PrintWriter

```
import java.io.File;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.BufferedWriter;
import java.io.IOException;

class AppendFileDemo2
{
   public static void main( String[] args )
   {
      try{
          File file =new File("C://myfile.txt");
                    if(!file.exists()){
                                   file.createNewFile();
                    }
                    FileWriter fw = new
FileWriter(file,true);
                    BufferedWriter bw = new
BufferedWriter(fw);
                    PrintWriter pw = new PrintWriter(bw);
          //This will add a new line to the file content
                    pw.println("");
/* Below three statements would add three
          * mentioned Strings to the file in new lines.
          */
                    pw.println("This is first line");
                    pw.println("This is the second line");
                    pw.println("This is third line");
                    pw.close();

                    System.out.println("Data successfully
appended at the end of file");

      }catch(IOException ioe){
                    System.out.println("Exception
occurred:");
                    ioe.printStackTrace();
      }
   }
}
```

How to delete file in Java – delete() Method

```
import java.io.File;
public class DeleteFileJavaDemo
{
   public static void main(String[] args)
   {
      try{
         //Specify the file name and path
                  File file = new File("C:\\myfile.txt");
         /*the delete() method returns true if the file is
          * deleted successfully else it returns false
          */
                  if(file.delete()){
                     System.out.println(file.getName() + " is deleted!");
         }else{
                     System.out.println("Delete failed: File didn't delete");
                   }
       }catch(Exception e){
           System.out.println("Exception occurred");
                    e.printStackTrace();
                 }
    }
}
```

How to rename file in Java – renameTo() method

```
import java.io.File;
public class RenameFileJavaDemo
{
    public static void main(String[] args)
    {
        //Old File
               File oldfile =new File("C:\\myfile.txt");
               //New File
               File newfile =new File("C:\\mynewfile.txt");
               /*renameTo() return boolean value
                * It return true if rename operation is
                * successful
                */
        boolean flag = oldfile.renameTo(newfile);
               if(flag){
                  System.out.println("File renamed successfully");
               }else{
                  System.out.println("Rename operation failed");
                }
    }
}
```

How to Compress a File in GZIP Format

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.zip.GZIPOutputStream;

public class GZipExample
{
  public static void main( String[] args )
  {
    GZipExample zipObj = new GZipExample();
    zipObj.gzipMyFile();
  }

  public void gzipMyFile(){
    byte[] buffer = new byte[1024];
    try{
      //Specify Name and Path of Output GZip file here
      GZIPOutputStream gos =
       new GZIPOutputStream(new
FileOutputStream("B://Java/Myfile.gz"));

      //Specify location of Input file here
      FileInputStream fis =
       new FileInputStream("B://Java/Myfile.txt");

      //Reading from input file and writing to output GZip file
      int length;
while ((length = fis.read(buffer)) > 0) {

        /* public void write(byte[] buf, int off, int len):
         * Writes array of bytes to the compressed output stream.
         * This method will block until all the bytes are written.
         * Parameters:
         * buf - the data to be written
         * off - the start offset of the data
         * len - the length of the data
         */
        gos.write(buffer, 0, length);
      }

      fis.close();

      /* public void finish(): Finishes writing compressed
       * data to the output stream without closing the
       * underlying stream.
       */
      gos.finish();
      gos.close();

      System.out.println("File Compressed!!");

    }catch(IOException ioe){
      ioe.printStackTrace();
    }
  }
}
```

How to Copy a File to another File in Java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class CopyExample
{
    public static void main(String[] args)
    {
                FileInputStream instream = null;
                FileOutputStream outstream = null;

                try{
                    File infile =new File("C:\\MyInputFile.txt");
                    File outfile =new
File("C:\\MyOutputFile.txt");

                    instream = new FileInputStream(infile);
                    outstream = new FileOutputStream(outfile);

                    byte[] buffer = new byte[1024];

                    int length;
                    /*copying the contents from input
stream to
                     * output stream using read and write
methods
                     */
 while ((length = instream.read(buffer)) > 0){
                            outstream.write(buffer, 0,
length);
                    }

                    //Closing the input/output file streams
                    instream.close();
                    outstream.close();

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

                }catch(IOException ioe){
                            ioe.printStackTrace();
                 }
    }
}
```

How to make a File Read Only in Java

```
import java.io.File;
import java.io.IOException;

public class ReadOnlyChangeExample
{

    public static void main(String[] args) throws IOException
    {
            File myfile = new File("C://Myfile.txt");
            //making the file read only
            boolean flag = myfile.setReadOnly();
            if (flag==true)
            {
              System.out.println("File successfully converted to Read only mode!!");
            }
            else
            {
              System.out.println("Unsuccessful Operation!!");
            }
    }
}
```

How to check if a File is hidden in Java

```
import java.io.File;
import java.io.IOException;

public class HiddenPropertyCheck
{

    public static void main(String[] args) throws IOException, SecurityException
    {
            // Provide the complete file path here
            File file = new File("c:/myfile.txt");

            if(file.isHidden()){
                    System.out.println("The specified file is hidden");
            }else{
                    System.out.println("The specified file is not hidden");
            }
    }
}
```