Different Formats For Reading And Writing

Input/output can be read/write as the following different formats:

#1) Reading File As Byte Stream

Here the data will be read in byte format. “**FileInputStream”** and “**FileOutputStream** “classes are used for reading the content as a byte. In this way, for every byte, the compiler will send a request to the OS.

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

public class ByteStreamsDemo {

public static void main(String[] args) throws IOException {

FileInputStream fin =null;
FileOutputStream fout =null;

try {

fin=new FileInputStream("input.txt");
fout=new FileOutputStream("out.txt");

int c;

while((c=fin.read() )!= -1)

{

fout.write(c);

}

} catch (FileNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}
```

```
finally {

if(fin!=null) {

fin.close();

}if(fout!=null) {

fout.close();

}
}
}

}
```

Wap to **copy the content of file from one file to another :**

Data.txt ⇒ output.txt

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
public class ByteStreamsDemo {
        public static void main(String[] args) throws IOException {
                FileInputStream fin =null;
                FileOutputStream fout =null;


                try {
                        fin=new FileInputStream("input.txt");
                        fout=new FileOutputStream("out.txt");
                        int c;
                        while((c=fin.read() )!= -1)
                   {
                                fout.write(c);

                        }
                } catch (FileNotFoundException e) {
                        // TODO Auto-generated catch block
```

Creating object for FileInputStream and FileOutputStream

Input.txt –source file to read

Reading the file Content and writing into out.txt. if the file reaches end, the read() method will return -1.

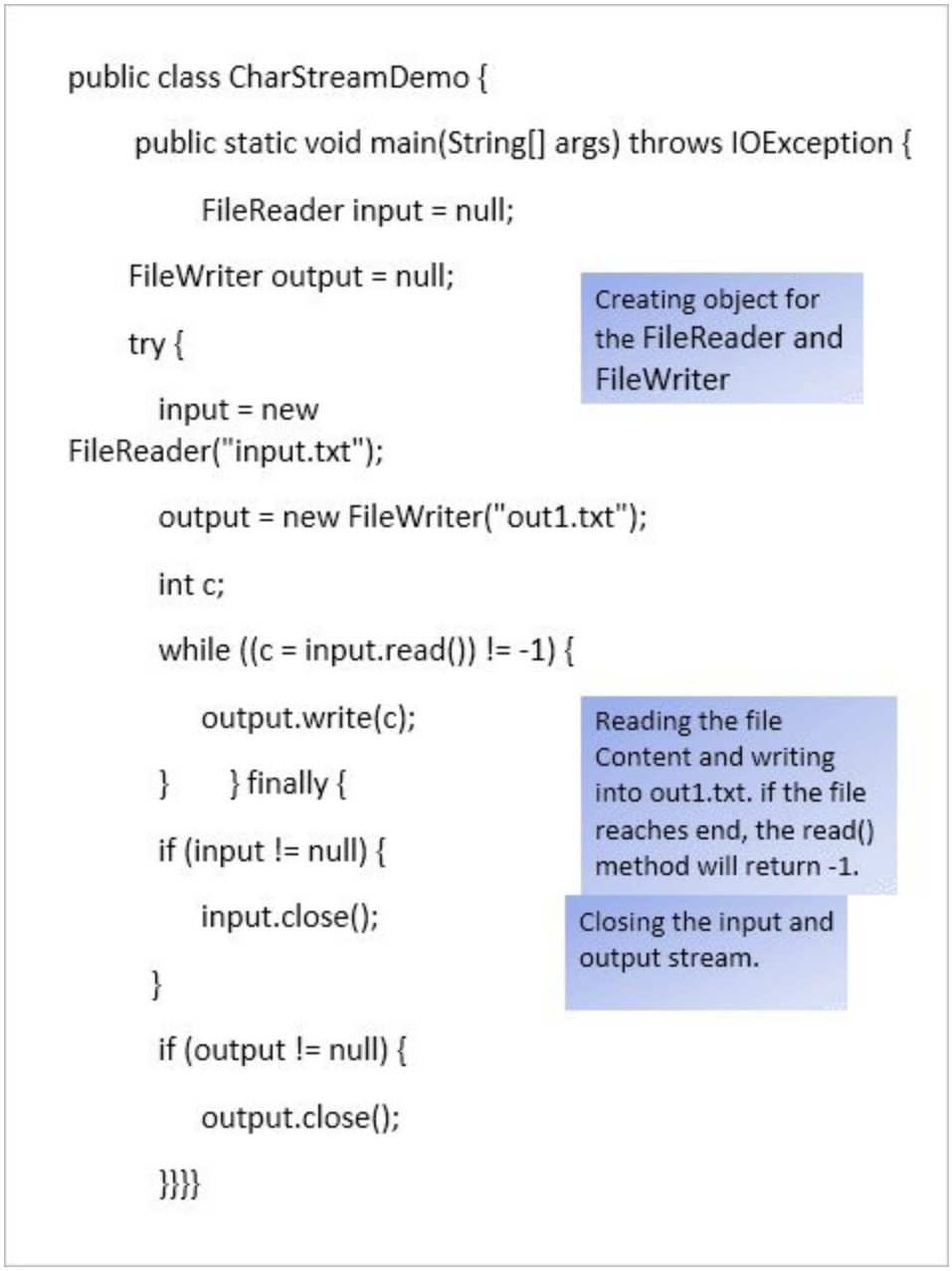
```
                e.printStackTrace();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            finally {
                if(fin!=null) {
                    fin.close();
                }if(fout!=null) {
                    fout.close();
                }
            }
        }
}
```

Closing the input and output stream.

#2) Reading File As Character Stream

In this way, the input stream will be read in character format. So, for each character, the compiler will send a request to the OS. “**FileReader”** and “**FileWriter”** classes are useful for reading the content as the character.

```
public class CharStreamDemo {

public static void main(String[] args) throws IOException {

FileReader input = null;
FileWriter output = null;
try {

input = new FileReader("input.txt");

output = new FileWriter("out1.txt");

int c;

while ((c = input.read()) != -1) {

output.write(c);

}  } finally {

if (input != null) {

input.close();

}

if (output != null) {

output.close();

}
}
}
}
```



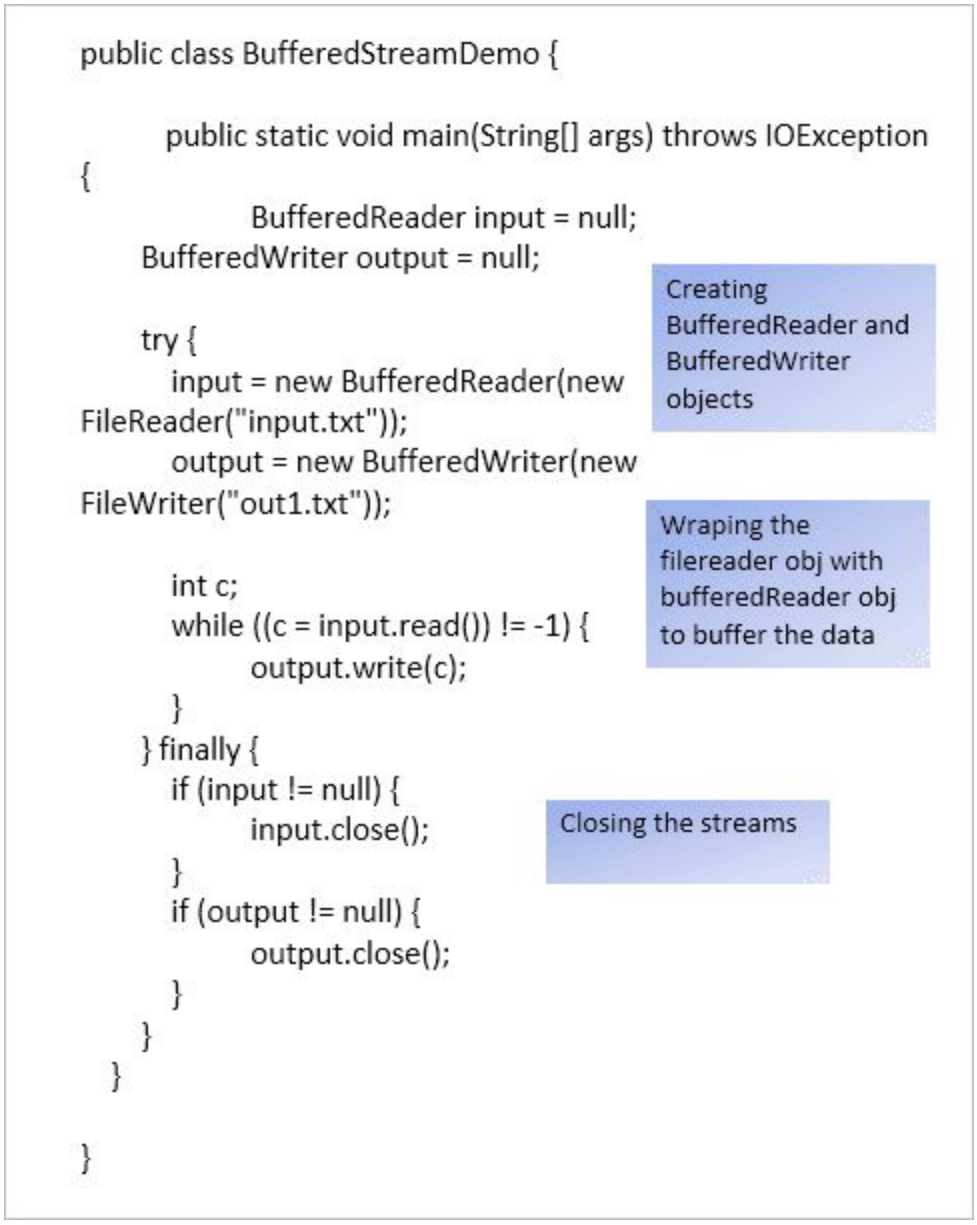
#3) Buffering The Input/Output Stream

When you use the FileInputStream or FileReader classes, for every read or write operation, a new request will be sent to OS. So, this may lead to performance issues. In order to avoid this BufferedInputStream or BufferedReader, classes are used to wrap unbuffered classes.

This reads the input stream when the buffer is empty.

Likewise, FileOutputStream or FileWriter, classes are wrapped with BufferedOutputStream or BufferedWriter to write the output once the buffer is full.

```
public class BufferedStreamDemo {

public static void main(String[] args) throws IOException
{

BufferedReader input = null;
BufferedWriter output = null;

try {
input = new BufferedReader(new FileReader("input.txt"));
output = new BufferedWriter(new FileWriter("out1.txt"));

int c;
while ((c = input.read()) != -1) {
output.write(c);
}
} finally {
if (input != null) {
input.close();
}
if (output != null) {
output.close();
}
}
}
}
```



#4) Reading As Data Stream

In this method, the DataInputStream or DataOutputStream, classes are used to read and write the content as primitive data types such as boolean, char, byte, short, int, long, float, double and String. Mostly the DataInputStream and DataOutputStream will be used together.

```
public class DataInputOutputStreamDemo {
public static void main(String[] args) {
File file = new File("read.bin");

FileOutputStream fos = null;
DataOutputStream dos = null;
try {

fos=new FileOutputStream(file);
dos=new DataOutputStream(fos);
dos.writeInt(50244);
dos.writeDouble(400.253);
dos.writeChar('d');
dos.flush();
} catch (IOException e) {
e.printStackTrace();
}finally {
try {

if(fos!=null){
fos.close();
}
if(dos!=null){
dos.close();
}
} catch (Exception e) {

e.printStackTrace();
}
}
```

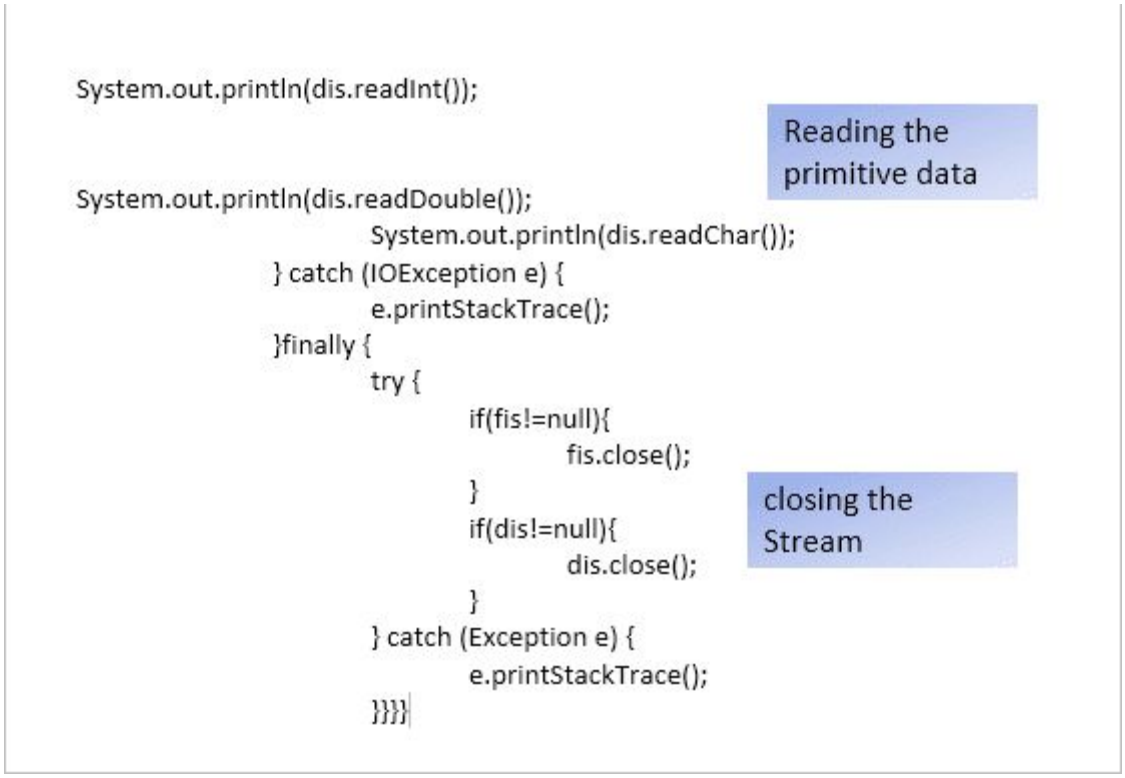
```
/*Reading operation */
FileInputStream fis = null;
DataInputStream dis = null;

try {

fis = new FileInputStream(file);
```

```
dis = new DataInputStream(fis);
System.out.println(dis.readInt());

System.out.println(dis.readDouble());
System.out.println(dis.readChar());
} catch (IOException e) {
e.printStackTrace();
}finally {
try {
if(fis!=null){
fis.close();
}
if(dis!=null){
dis.close();
}
} catch (Exception e) {
e.printStackTrace();
}

}
}
}
```

```
public class DataInputOutputStreamDemo {
	public static void main(String[] args) {
		File file = new File("read.bin");


		FileOutputStream fos = null;
		DataOutputStream dos = null;
		try {
			fos=new FileOutputStream(file);
			dos=new DataOutputStream(fos);
			dos.writeInt(50244);
			dos.writeDouble(400.253);
			dos.writeChar('d');
			dos.flush();
		} catch (IOException e) {
			e.printStackTrace();
		}finally {
			try {
				if(fos!=null){
					fos.close();
				}
				if(dos!=null){
					dos.close();
				}
			} catch (Exception e) {
				e.printStackTrace();
			}
		}


		/*Reading operation */
		FileInputStream fis = null;
		DataInputStream dis = null;

		try {
			fis = new FileInputStream(file);
			dis = new DataInputStream(fis);
```

Creating a file

Writing the primitive data



#5) Reading As Object Stream

ObjectInputStream/ ObjectOutputStream, classes are useful to write objects in a file and read the objects from the file. For storing the object in a file, the class should implement the Serializable interface.

```
public class ObjectStreamDemo implements Serializable {

int age ;
String name;
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public int getAge() {
return age;
}
public void setAge(int age) {
this.age = age;
```

} }

```
public class ObjectStreamDemo implements Serializable {

 int age ;
 String name;
public String getName() {
        return name;
}
public void setName(String name) {
        this.name = name;
}
public int getAge() {
        return age;
}
public void setAge(int age) {
        this.age = age;
} }
```

This class is used to save values to the object

We are going to create an object for this “ObjectStreamDemo” class and we will write that object into a file and read the same object from that file.

```
public class ObjectStreamDemoTest {

public static void main(String[] args) {
// TODO Auto-generated method stub

ObjectStreamDemo obj=new ObjectStreamDemo();

obj.setAge(32);
obj.setName("bob");
try {
FileOutputStream fos =
new FileOutputStream("t.tmp");
ObjectOutputStream oos = new
ObjectOutputStream(fos);
oos.writeObject(obj);
oos.close();

} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

FileInputStream fis;
ObjectInputStream ois;
try {

fis = new FileInputStream("t.tmp");
ois = new ObjectInputStream(fis);
ObjectStreamDemo obj1
= (ObjectStreamDemo)ois.readObject();

System.out.println(obj1.name);
System.out.println(obj1.age);
} catch (FileNotFoundException e) {
// TODO Auto-generated catch block

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

```
catch(Exception e) {
e.printStackTrace();

}
}
}
```

```
public class ObjectStreamDemoTest {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		ObjectStreamDemo obj=new ObjectStreamDemo();

		obj.setAge(32);
		obj.setName("bob");
		 try {
		FileOutputStream fos = new
FileOutputStream("t.tmp");
	ObjectOutputStream oos = new ObjectOutputStream(fos);
	oos.writeObject(obj);
	oos.close();

		} catch (IOException e) {
			// TODO Auto-generated
catch block
			e.printStackTrace();
		}

		 FileInputStream fis;
		 ObjectInputStream ois;
		try {
			fis = new FileInputStream("t.tmp");
			ois = new ObjectInputStream(fis);
			ObjectStreamDemo obj1 =
(ObjectStreamDemo)ois.readObject();

			System.out.println(obj1.name);
			System.out.println(obj1.age);
		} catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (IOException e) {
```

Creating instance of ObjectStreamDemo values to the object and setting the values

Writing the object into the t.tmp file

Reading the object from the t.tmp file

```
                e.printStackTrace();
            }

            catch(Exception e) {
                e.printStackTrace();

            }

        }

}
```

File I/O Operations

File class is useful for doing file operations.

Some of the file operations performed using File class include:

1. Create a file
2. Check if the file is existing
3. Get the path of the file
4. Write the file
5. Read the file
6. Delete a file and rename the file
7. Check the file permissions and change the file permissions
8. Move the file from one directory to another location

Demo Program to create, read, and write the file:

```
public class CreateFileDemo {

public static void main(String[] args) throws IOException {

File newfile=new File("created.txt");

if(!newfile.exists()) {
newfile.createNewFile();

System.out.println("file not exist");

}

try {

String FILENAME="created.txt";
String content="hi how are u";

FileWriter fwt = new FileWriter(FILENAME);
BufferedWriter bwt = new BufferedWriter(fwt);

bwt.write(content);

System.out.println("writing completed ...");

bwt.close();

} catch (IOException e) {

e.printStackTrace();

}
}
}
```

```
public class CreateFileDemo {

	public static void main(String[] args) throws IOException {

		File newfile=new File("created.txt");

		if(!newfile.exists()) {
	newfile.createNewFile();

		System.out.println("file not
exist");

		}

	try {

		String FILENAME="created.txt";
String content="hi how are u";

			FileWriter fwt = new FileWriter(FILENAME);

			BufferedWriter bwt = new BufferedWriter(fwt);

				bwt.write(content);

	System.out.println("writing completed ...");

				bwt.close();

			} catch (IOException e) {

				e.printStackTrace();

			}}

		}
```

Creating New file created.txt

Checking if file exists

Key points to be noted:

- A stream is a logical representation of the flow of data.
- You can read/write data in a different format like byte, character, object, primitive data type.
- File class is used to create a file, delete the file and Move or copy or rename the file.

- BufferedInputStream or BufferedOutputStream is used to improve the performance by buffering the data.

Conclusion

Java has a java.lang package that provides the standard input and output facilities using the System class.

Apart from the streams, System.in and System.out that are used for standard input and output respectively, there are also other methods like BufferedReader, console class and scanner class that is used to read input from the user.

System.out stream uses “PrintStream” class function, print and println to display the output. These are the functions that are used to display the output without formatting. Another function “printf” which is similar to printf function in C/C++ is also used in Java for formatted output.