Object Oriented Programming using Java

Prepared By:
Suyel, PhD
Assistant Professor
Dept. of CSE, NIT Patna

Outline

1. Byte Stream
2. Character Stream

Introduction

- Data stored in variables and arrays are temporary.
- Data are lost, when a local variable goes out of scope or when the program is terminated.
- For long-term use of data, we use files. It is actually a collection of data.
- Data maintained in files is persistent data.
- Computers store files on secondary storage devices.
- Java provides java.io package in which each class represents one source or device.

Introduction (Cont...)

- In Java, the sources or destinations of any program are defined very broadly. For example, a network connection, memory buffer or disk file can be manipulated by the Java I/O classes.
- **A stream is an abstraction or a logical entity that either produces or consumes information.**
- A stream is linked to a physical device by the Java I/O system.
- All streams behave in the same manner, even if the actual physical devices they are linked to differ.
- Java defines two types of streams: byte and character.

Introduction (Cont...)

- **Java I/O stream or File Handling or File I/O** is the flow of data that we can read from or we can write to.
- It is used to perform read and write operations in a file permanently.
- Java.io package provides classes for system input and output through files, network streams, memory buffers, etc.
- JVM automatically initiates some I/O streams.
- **System** class of **java.lang** has three predefined stream variables: **in, out** and **err.**
  - **In** reference refers to the default input device i.e. keyboard.
  - **Out** and **err** refers to the default output device i.e. console.

I/O Classes and Interfaces

- **Some I/O Classes**

BufferedInputStream	FileWriter	PipedOutputStream
BufferedOutputStream	FilterInputStream	PipedReader
BufferedReader	FilterOutputStream	PipedWriter
BufferedWriter	FilterReader	PrintStream
.	.	.

- **I/O Interface**

Closeable	FileFilter	ObjectInputValidation
DataInput	FilenameFilter	ObjectOutput
DataOutput	Flushable	ObjectStreamConstants
Externalizable	ObjectInput	Serializable

File Class

- ❑ Although most of the classes defined by java.io operate on streams, the File class does not.

- ❑ It deals directly with files and file system.

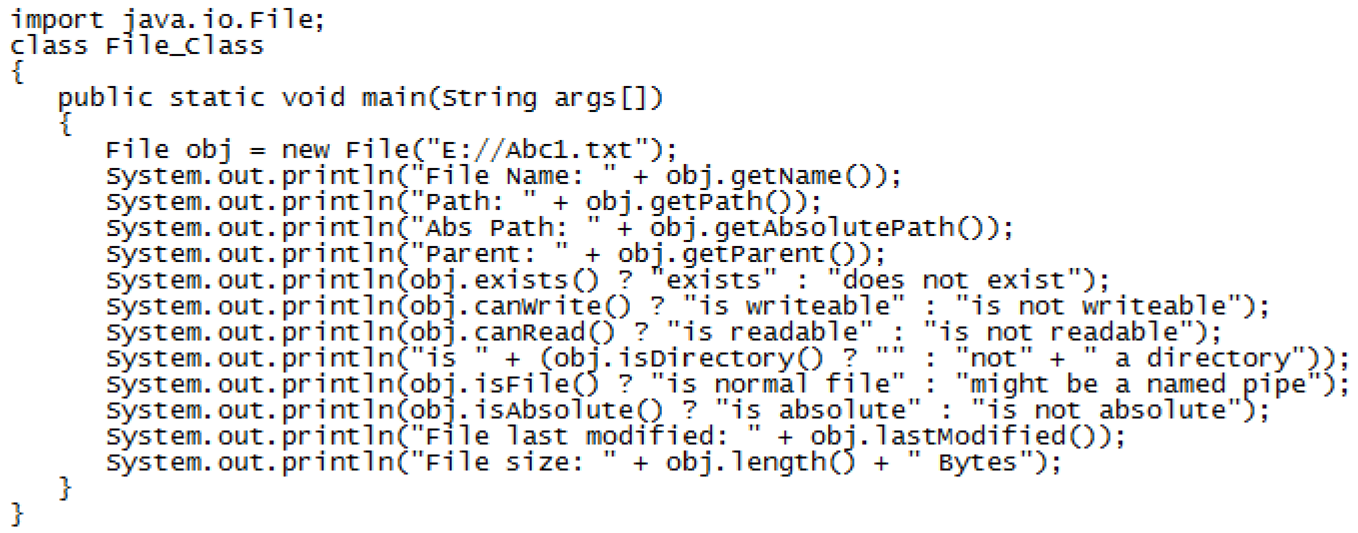
- ❑ File class does not specify how information is retrieved from or stored in files. It describes the properties of a file.

- ❑ File object is used to obtain or manipulate the information associated with a disk file.

File Class (Cont...)

- The following **constructors** can be used to create File objects:
  - File(String *directoryPath*) //File f1 = new File("/");
  - File(String *directoryPath*, String *filename*) //new File("/",“my.txt");
  - File(File *dirObj*, String *filename*) //File f2 = new File(f1,“my.txt");
  - File(URI *uriObj*) //URI is a sequence of characters to identify a resource
- There are many **methods** in the File class. **Some** are:
  - getName( ): It gives the name of the file.
  - getParent( ): It gives the name of the parent directory.
  - exists( ): Returns true, if the file exists, and false, if it does not.

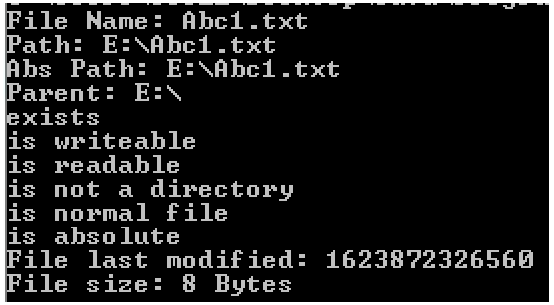
File Class (Cont...)



```
import java.io.File;
class File_Class
{
   public static void main(String args[])
   {
      File obj = new File("E://Abc1.txt");
      System.out.println("File Name: " + obj.getName());
      System.out.println("Path: " + obj.getPath());
      System.out.println("Abs Path: " + obj.getAbsolutePath());
      System.out.println("Parent: " + obj.getParent());
      System.out.println(obj.exists() ? "exists" : "does not exist");
      System.out.println(obj.canWrite() ? "is writeable" : "is not writeable");
      System.out.println(obj.canRead() ? "is readable" : "is not readable");
      System.out.println("is " + (obj.isDirectory() ? "" : "not" + " a directory"));
      System.out.println(obj.isFile() ? "is normal file" : "might be a named pipe");
      System.out.println(obj.isAbsolute() ? "is absolute" : "is not absolute");
      System.out.println("File last modified: " + obj.lastModified());
      System.out.println("File size: " + obj.length() + " Bytes");
   }
}
```

File Class (Cont...)

- **Output:**

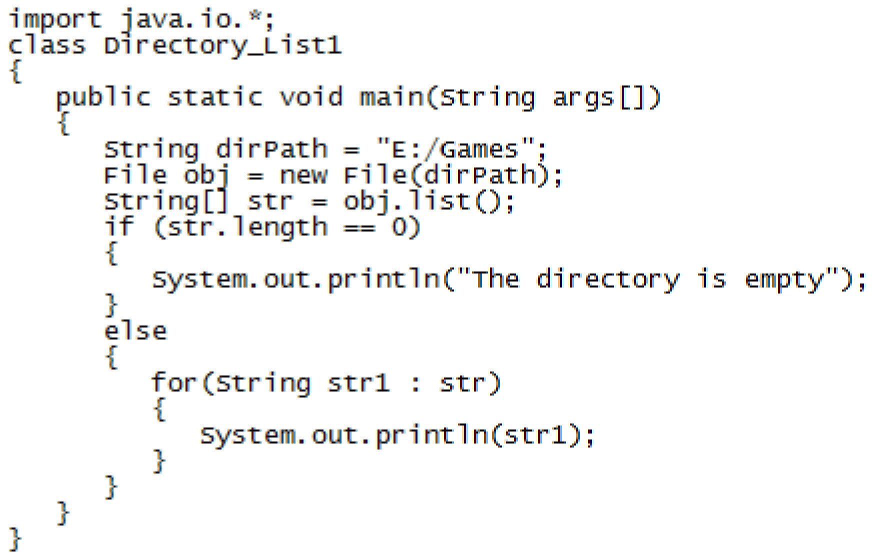


```
File Name: Abc1.txt
Path: E:\Abc1.txt
Abs Path: E:\Abc1.txt
Parent: E:\
exists
is writeable
is readable
is not a directory
is normal file
is absolute
File last modified: 1623872326560
File size: 8 Bytes
```

Directories

- The directory is a file that contains a list of other files and directories.
- There are some useful methods of directory:
  - isDirectory( ): When we create a File object and it is a directory, the isDirectory( ) method will return true.
  - list( ): We can call list( ) to extract the list files inside a directory. There are two ways to use list:
    - String[ ] list( )
    - String[ ] list(FilenameFilter *FFObj*) //*FFObj* is an object of a class that implements the FilenameFilter interface.

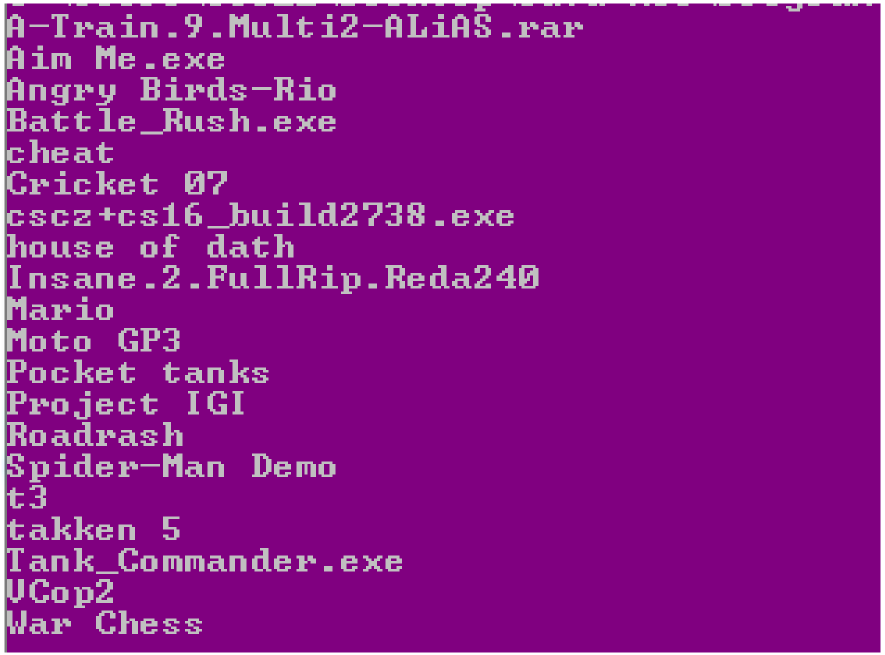
Directories (Cont...)



```
import java.io.*;
class Directory_List1
{
   public static void main(String args[])
   {
      String dirPath = "E:/Games";
      File obj = new File(dirPath);
      String[] str = obj.list();
      if (str.length == 0)
      {
         System.out.println("The directory is empty");
      }
      else
      {
         for(String str1 : str)
         {
            System.out.println(str1);
         }
      }
   }
}
```

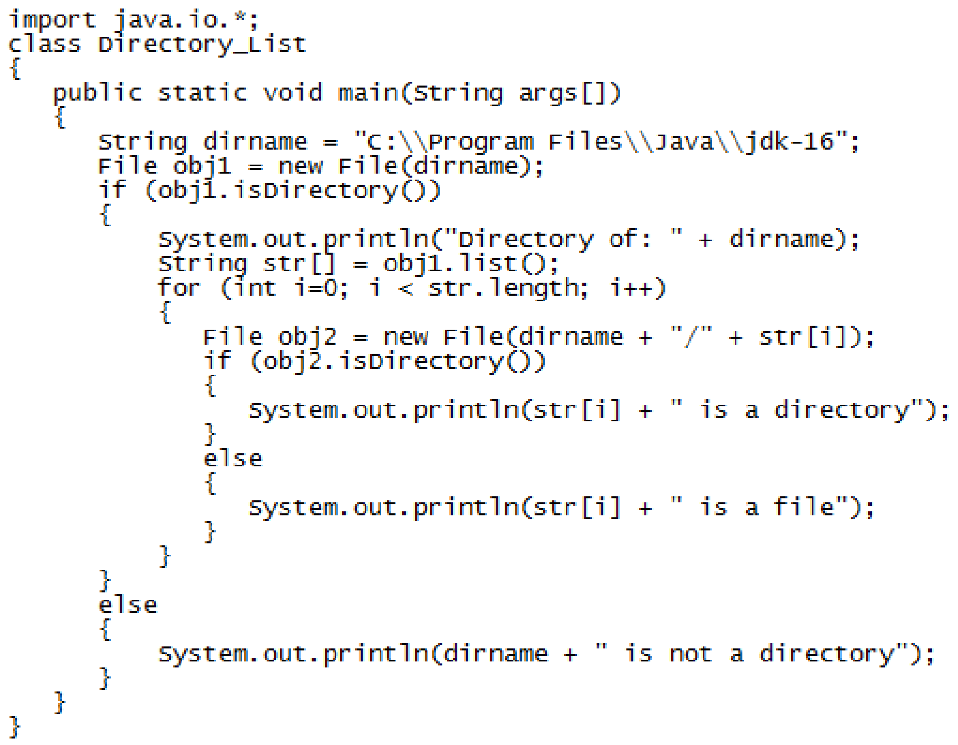
Directories (Cont...)

- **Output:**



```
A-Train.9.Multi2-ALiAS.rar
Aim Me.exe
Angry Birds-Rio
Battle_Rush.exe
cheat
Cricket 07
cscz+cs16_build2738.exe
house of dath
Insane.2.FullRip.Reda240
Mario
Moto GP3
Pocket tanks
Project IGI
Roadrash
Spider-Man Demo
t3
takken 5
Tank_Commander.exe
VCop2
War Chess
```

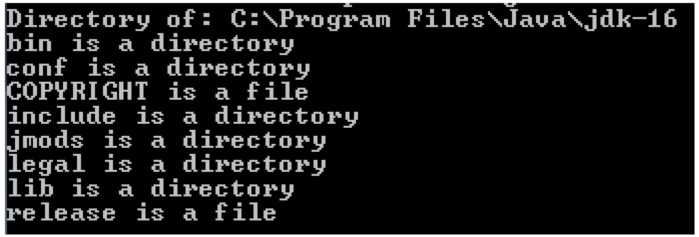
Directories (Cont...)



```
import java.io.*;
class Directory_List
{
   public static void main(String args[])
   {
      String dirname = "C:\\Program Files\\Java\\jdk-16";
      File obj1 = new File(dirname);
      if (obj1.isDirectory())
      {
          System.out.println("Directory of: " + dirname);
          String str[] = obj1.list();
          for (int i=0; i < str.length; i++)
          {
             File obj2 = new File(dirname + "/" + str[i]);
             if (obj2.isDirectory())
             {
                System.out.println(str[i] + " is a directory");
             }
             else
             {
                System.out.println(str[i] + " is a file");
             }
          }
      }
      else
      {
          System.out.println(dirname + " is not a directory");
      }
   }
}
```

Directories (Cont...)

- **Output**



```
Directory of: C:\Program Files\Java\jdk-16
bin is a directory
conf is a directory
COPYRIGHT is a file
include is a directory
jmods is a directory
legal is a directory
lib is a directory
release is a file
```

FilenameFilter

- We can get the number of files returned by the list( ) to include only those files that match a certain filename pattern or filter.

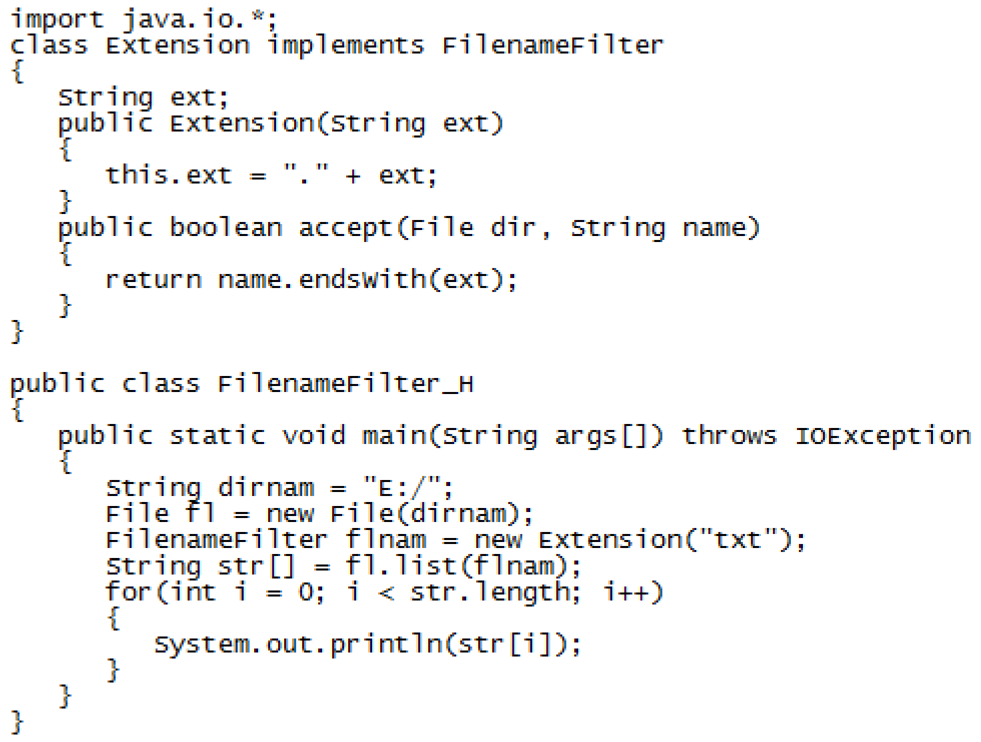
  String[ ] list(FilenameFilter *FFObj*) //Here, FFObj is an object of a class that implements the FilenameFilter interface.

- FilenameFilter defines only a single method i.e. accept( ) and this method is called once for each file in a list.

  boolean accept(File *directory*, String *filename*)

- The accept( ) method returns true for files in the directory specified by directory that should be included in the list.

FilenameFilter (Cont...)

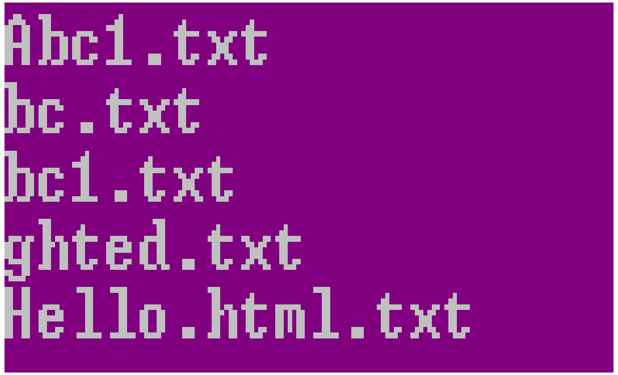


```
import java.io.*;
class Extension implements FilenameFilter
{
   String ext;
   public Extension(String ext)
   {
      this.ext = "." + ext;
   }
   public boolean accept(File dir, String name)
   {
      return name.endsWith(ext);
   }
}

public class FilenameFilter_H
{
   public static void main(String args[]) throws IOException
   {
      String dirnam = "E:/";
      File fl = new File(dirnam);
      FilenameFilter flnam = new Extension("txt");
      String str[] = fl.list(flnam);
      for(int i = 0; i < str.length; i++)
      {
         System.out.println(str[i]);
      }
   }
}
```

FilenameFilter (Cont...)

- Output



```
Abc1.txt
bc.txt
bc1.txt
ghted.txt
Hello.html.txt
```

Stream Classes

- There are four abstract classes:
  - InputStream
  - OutputStream
  - Reader
  - Writer
- The above mentioned classes provide basic functionalities.
- InputStream and OutputStream are designed for **byte streams.**
- Reader and Writer are designed for **character streams**.

Byte Stream

- Java byte streams are used to perform input and output of byte oriented.
- It is important in many applications.
- Byte stream classes are topped by:
  - **InputStream:** It implements the **Closeable** interface. Most of the methods of this class throw an **IOException**.
  - **OutputStream:** It implements the **Closeable** and **Flushable** interfaces. Most of the methods in this class return **void** and throw an **IOException.**
- There are many classes in byte stream. Most frequently used classes are: **FileInputStream** and **FileOutputStream**.

FileInputStream

- The FileInputStream class creates an InputStream that we can use to read bytes from a file.
- Its two most common **constructors** are shown below:

  FileInputStream(String *filepath*)   //filepath is the full path name of a file

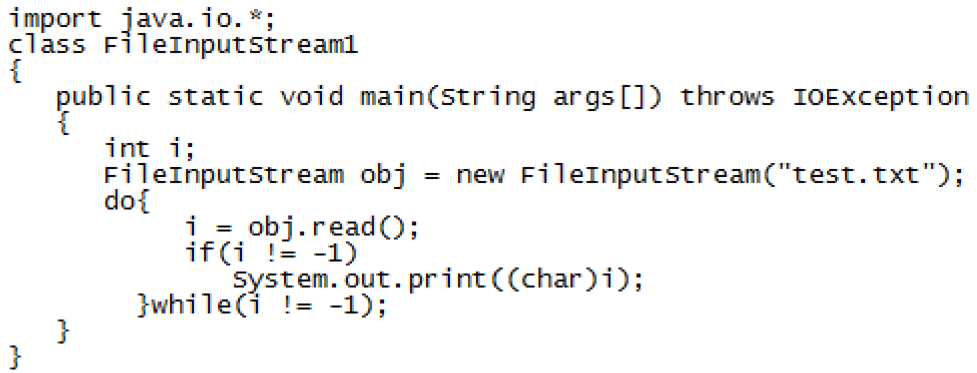
  FileInputStream(File *fileObj*)   //fileObj is a File object
- Example:

  FileInputStream f0 = new FileInputStream(“../Abc.txt");

  File f = new File(“../Abc.txt");

  FileInputStream f1 = new FileInputStream(f);
- If FileInputStream is created, it is also opened for reading.

FileInputStream (Cont…)



```
import java.io.*;
class FileInputStream1
{
   public static void main(String args[]) throws IOException
   {
      int i;
      FileInputStream obj = new FileInputStream("test.txt");
      do{
           i = obj.read();
           if(i != -1)
              System.out.print((char)i);
        }while(i != -1);
   }
}
```

FileInputStream (Cont…)

- **Output**

```
I want to print this line to my file
```

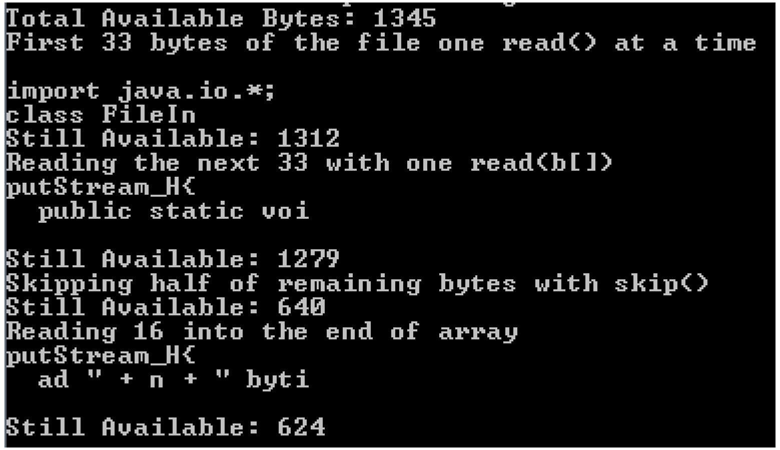
The above line is the content of the "text.txt" file.

FileInputStream (Cont…)

```java
import java.io.*;
class FileInputStream_H{
  public static void main(String args[]) throws IOException
  {
     int size;
     InputStream f = new FileInputStream("FileInputStream_H.java");
     System.out.println("Total Available Bytes: " + (size = f.available()));
     int n = size/40;
     System.out.println("First " + n + " bytes of the file one read() at a time");
     for (int i=0; i < n; i++){
        System.out.print((char) f.read());
     }
     System.out.println("\nStill Available: " + f.available());
     System.out.println("Reading the next " + n + " with one read(b[])");
     byte b[] = new byte[n];
     if (f.read(b) != n) {
        System.err.println("couldn't read " + n + " bytes.");
     }
     System.out.println(new String(b, 0, n));
     System.out.println("\nStill Available: " + (size = f.available()));
     System.out.println("Skipping half of remaining bytes with skip()");
     f.skip(size/2);
     System.out.println("Still Available: " + f.available());
     System.out.println("Reading " + n/2 + " into the end of array");
     if (f.read(b, n/2, n/2) != n/2) {
        System.err.println("couldn't read " + n/2 + " bytes.");
     }
     System.out.println(new String(b, 0, b.length));
     System.out.println("\nStill Available: " + f.available());
     f.close();
  }
}
```

FileInputStream (Cont…)

- **Output**



```
Total Available Bytes: 1345
First 33 bytes of the file one read() at a time

import java.io.*;
class FileIn
Still Available: 1312
Reading the next 33 with one read(b[])
putStream_H{
  public static voi

Still Available: 1279
Skipping half of remaining bytes with skip()
Still Available: 640
Reading 16 into the end of array
putStream_H{
  ad " + n + " byti

Still Available: 624
```

FileOputStream

- FileOutputStream creates an OutputStream that we can use to write bytes to a file.
- We can write byte oriented as well as character oriented data through FileOutputStream class. But, it is preferred to use FileWriter for character oriented data.
- Its most commonly used **constructors** are shown below:
  - FileOutputStream(String *filePath)*
  - FileOutputStream(File *fileObj)*
  - FileOutputStream(String *filePath*, boolean *append)*
  - FileOutputStream(File *fileObj*, boolean *append)*

FileOputStream (Cont…)

```
import java.io.*;
public class FileOutputStream1
{
    public static void main(String args[]) throws IOException
    {
        int i = 0;
        FileOutputStream obj = new FileOutputStream("E:\\Abc1.txt");
        String str = "Hello, how are you?";
        char c[] = str.toCharArray();
        for(i = 0; i < str.length(); i++)
            obj.write(c[i]);
        obj.close();
        System.out.println("The Program has been successfully executed");
    }
}
```

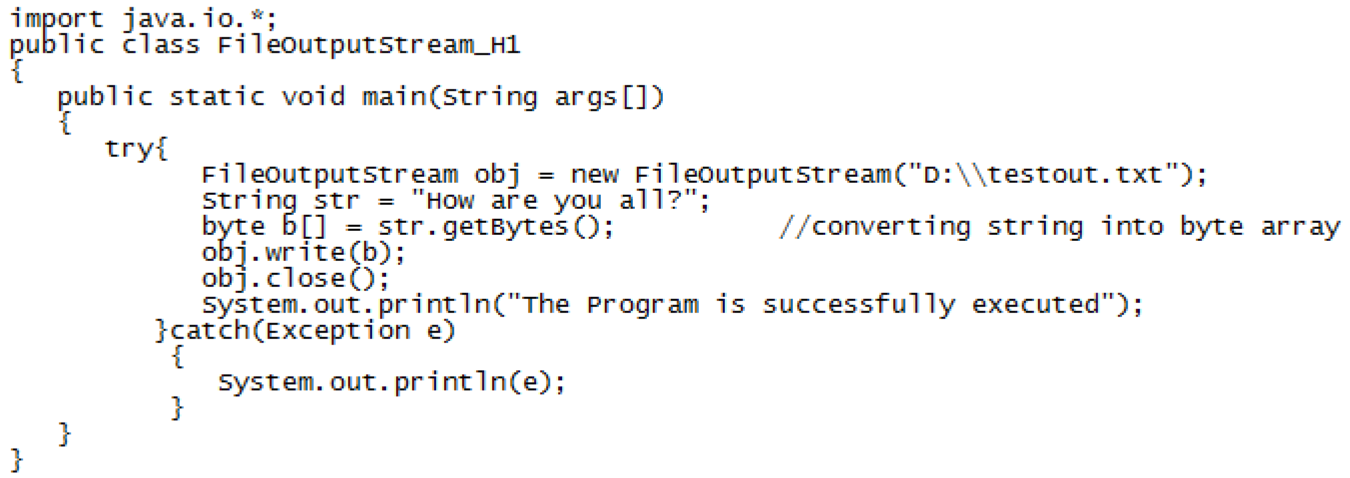
FileOputStream (Cont…)

- **Output**

  Program has been successfully executed

  “How are you all?” Will be stored in the file

FileOputStream (Cont…)



```
import java.io.*;
public class FileOutputStream_H1
{
   public static void main(String args[])
   {
      try{
            FileOutputStream obj = new FileOutputStream("D:\\testout.txt");
            String str = "How are you all?";
            byte b[] = str.getBytes();          //converting string into byte array
            obj.write(b);
            obj.close();
            System.out.println("The Program is successfully executed");
         }catch(Exception e)
          {
             System.out.println(e);
          }
   }
}
```

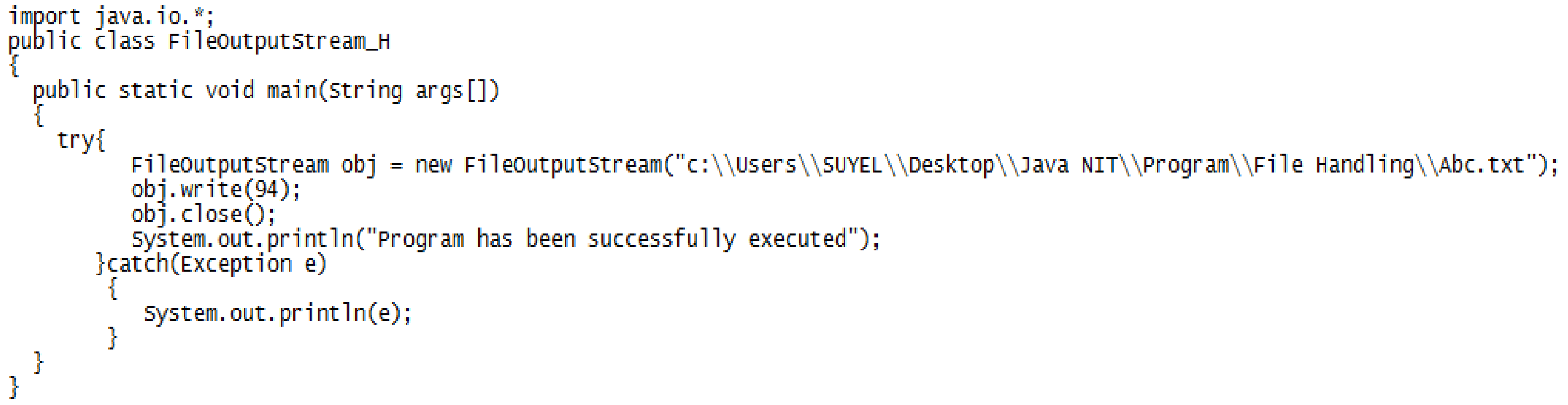
FileOputStream (Cont…)

- **Output**

  The program is successfully executed

  In the testout.txt file, “How are you all” is saved.

FileOputStream (Cont…)



```
import java.io.*;
public class FileOutputStream_H
{
  public static void main(String args[])
  {
    try{
          FileOutputStream obj = new FileOutputStream("c:\\Users\\SUYEL\\Desktop\\Java NIT\\Program\\File Handling\\Abc.txt");
          obj.write(94);
          obj.close();
          System.out.println("Program has been successfully executed");
       }catch(Exception e)
        {
           System.out.println(e);
        }
  }
}
```

FileOputStream (Cont…)

- **Output**

  Program has been successfully executed

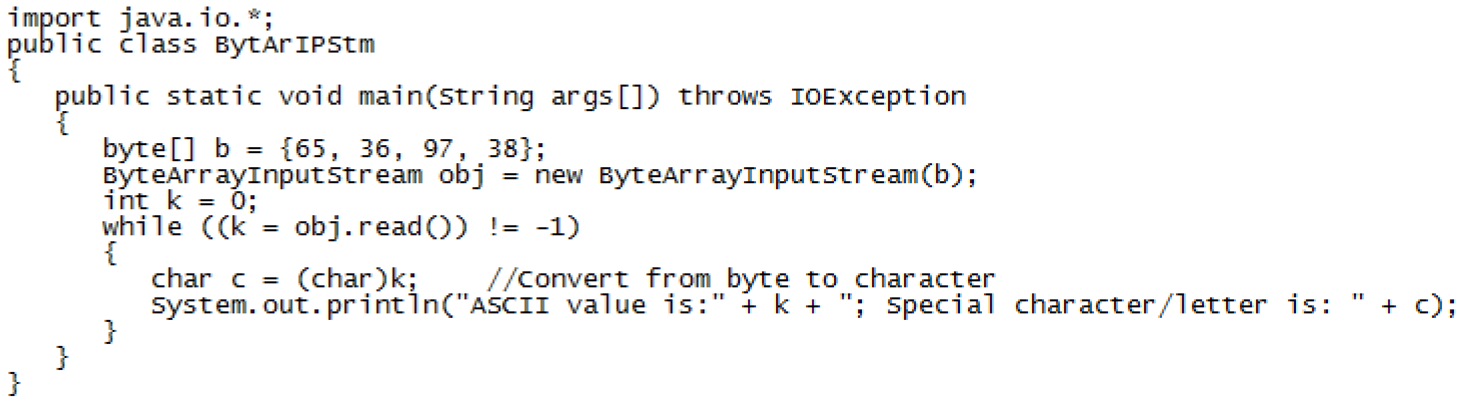
  According to ASCII value, data will be stored in the file.

ByteArrayInputStream

- ByteArrayInputStream is an implementation of an input stream that uses a byte array as the source.
- This class has two **constructors**:
  - ByteArrayInputStream(byte *array*[ ])
  - ByteArrayInputStream(byte *array*[ ], int *start*, int *numBytes*)

  Here, array [] is the input source. The 2nd constructor creates an InputStream from a subset of the byte array that starts with the index specified by start and is numBytes long.
- If mark( ) has not been called, then reset( ) sets the stream pointer to the start of the stream.

ByteArrayInputStream (Cont…)



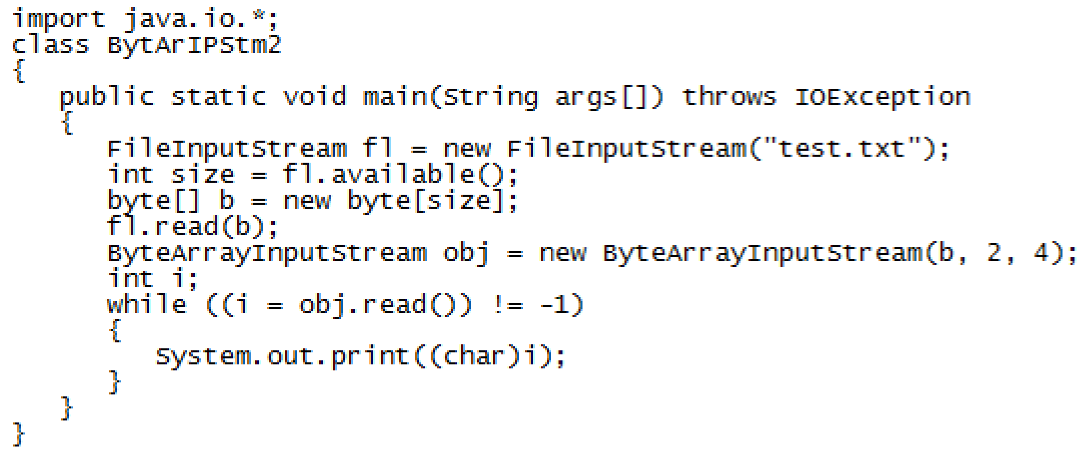
```
import java.io.*;
public class BytArIPStm
{
    public static void main(String args[]) throws IOException
    {
        byte[] b = {65, 36, 97, 38};
        ByteArrayInputStream obj = new ByteArrayInputStream(b);
        int k = 0;
        while ((k = obj.read()) != -1)
        {
            char c = (char)k;    //Convert from byte to character
            System.out.println("ASCII value is:" + k + "; Special character/letter is: " + c);
        }
    }
}
```

ByteArrayInputStream (Cont…)

- **Output**

```
ASCII value is: 65; Special character/letter is: A
ASCII value is: 36; Special character/letter is: $
ASCII value is: 97; Special character/letter is: a
ASCII value is: 38; Special character/letter is: &
```

ByteArrayInputStream (Cont…)



```
import java.io.*;
class BytArIPStm2
{
   public static void main(String args[]) throws IOException
   {
      FileInputStream fl = new FileInputStream("test.txt");
      int size = fl.available();
      byte[] b = new byte[size];
      fl.read(b);
      ByteArrayInputStream obj = new ByteArrayInputStream(b, 2, 4);
      int i;
      while ((i = obj.read()) != -1)
      {
         System.out.print((char)i);
      }
   }
}
```

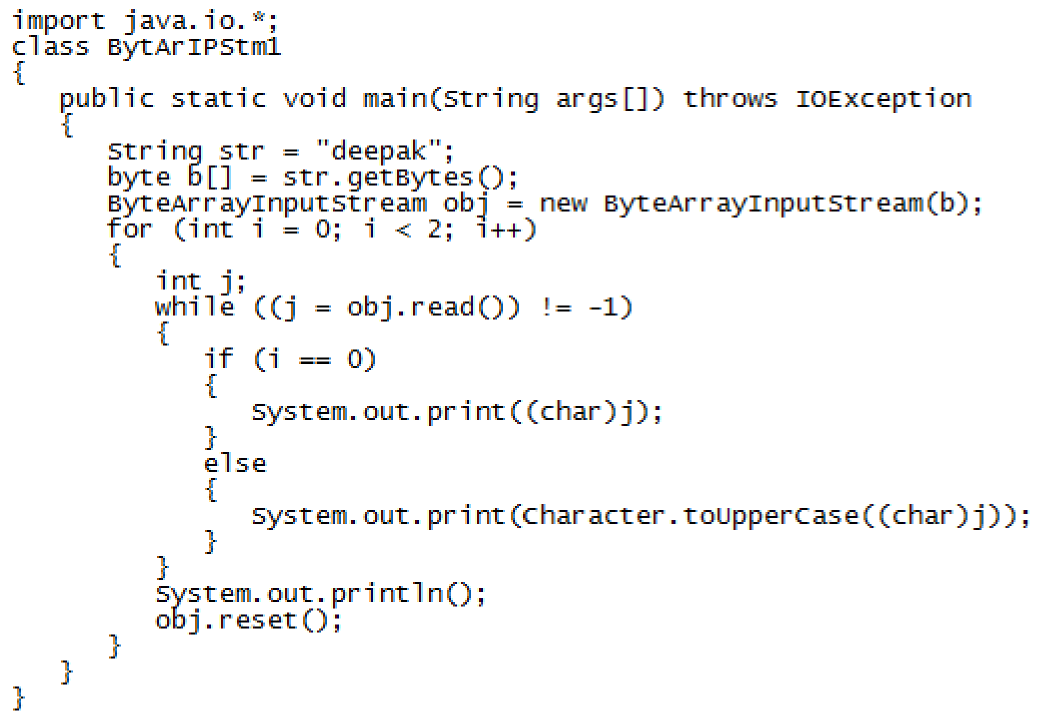
ByteArrayInputStream (Cont…)

- **Output**

```
want
```

The program will leave the contents of first 2 indexes of the array, and then, it will print the contents of 4 indexes.

ByteArrayInputStream (Cont…)



```
import java.io.*;
class BytArIPStm1
{
    public static void main(String args[]) throws IOException
    {
        String str = "deepak";
        byte b[] = str.getBytes();
        ByteArrayInputStream obj = new ByteArrayInputStream(b);
        for (int i = 0; i < 2; i++)
        {
            int j;
            while ((j = obj.read()) != -1)
            {
                if (i == 0)
                {
                    System.out.print((char)j);
                }
                else
                {
                    System.out.print(Character.toUpperCase((char)j));
                }
            }
            System.out.println();
            obj.reset();
        }
    }
}
```

ByteArrayInputStream (Cont…)

- **Output**

```
deepak
DEEPAK
```

ByteArrayOutputStream

- ByteArrayOutputStream writes the content of a byte array to its own internal buffer. Next, all the bytes in the internal buffer are written to a file using an output stream.
- It is an implementation of an output stream. Constructors:

  ByteArrayOutputStream( )

  ByteArrayOutputStream(int *numBytes*)

  In the 1st form, a buffer of 32 bytes is created. In the 2nd form, a buffer is created with a size equal to numBytes.
- The buffer and the number of bytes in the buffer are held in the protected buf and count fields of ByteArrayOutputStream, respectively.

ByteArrayOutputStream (Cont…)

```
import java.io.*;
class ByteArrayOutputStream_H1
{
   public static void main(String args[]) throws IOException
   {
      FileOutputStream fout1 = new FileOutputStream("Z1.txt");
      FileOutputStream fout2 = new FileOutputStream("Z2.txt");
      ByteArrayOutputStream bout = new ByteArrayOutputStream();
      bout.write(54);
      bout.writeTo(fout1);
      bout.writeTo(fout2);
      bout.flush();
      bout.close();
      System.out.println("First Part of the Program has Executed");

      FileOutputStream fout3 = new FileOutputStream("Z3.txt");
      FileOutputStream fout4 = new FileOutputStream("Z4.txt");
      ByteArrayOutputStream bout1 = new ByteArrayOutputStream();
      String str = "Hello how are you?";
      byte b[] = str.getBytes();
      bout1.write(b);
      bout1.writeTo(fout3);
      bout1.writeTo(fout4);
      bout1.flush();
      bout1.close();
      System.out.println("Second Part of the Program has also Executed");
   }
}
```

ByteArrayOutputStream (Cont…)

- **Output**

```
First Part of the Program has Executed
Second Part of the Program has also Executed
```

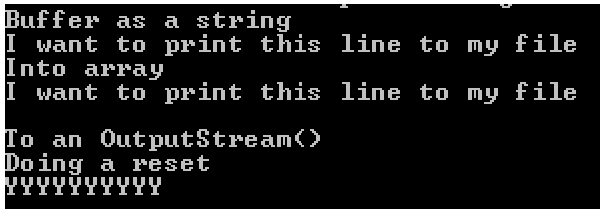
ASCII value of 54 i.e. 6 will be stored in Z1.txt and Z2.txt files and "Hello how are you?" will be stored in Z3.txt and Z4.txt files.

ByteArrayOutputStream (Cont…)

```
import java.io.*;
class ByteArrayOutputStream_H
{
   public static void main(String args[]) throws IOException
   {
      ByteArrayOutputStream obj = new ByteArrayOutputStream();
      String str = "I want to print this line to my file";
      byte b1[] = str.getBytes();
      obj.write(b1);
      System.out.println("Buffer as a string");
      System.out.println(obj.toString());
      System.out.println("Into array");
      byte b2[] = obj.toByteArray();
      for (int i = 0; i < b2.length; i++)
      {
         System.out.print((char)b2[i]);
      }
      System.out.println("\n\nTo an OutputStream()");
      OutputStream obj1 = new FileOutputStream("test.txt");
      obj.writeTo(obj1);
      obj1.close();
      System.out.println("Doing a reset");
      obj.reset();
      for (int i = 0; i < 10; i++)
         obj.write('Y');
      System.out.println(obj.toString());
   }
}
```

ByteArrayOutputStream (Cont…)

- **Output**



```
Buffer as a string
I want to print this line to my file
Into array
I want to print this line to my file

To an OutputStream()
Doing a reset
YYYYYYYYYY
```

Buffered Byte Streams

- For the byte-oriented streams, a buffered stream extends a filtered stream class by attaching a memory buffer to the I/O streams.
- Buffer allows us to do I/O operations on more than a byte at a time. Thus, it increases performance.
- There are mainly two buffered byte stream classes:
  - BufferedInputStream
  - BufferedOutputStream

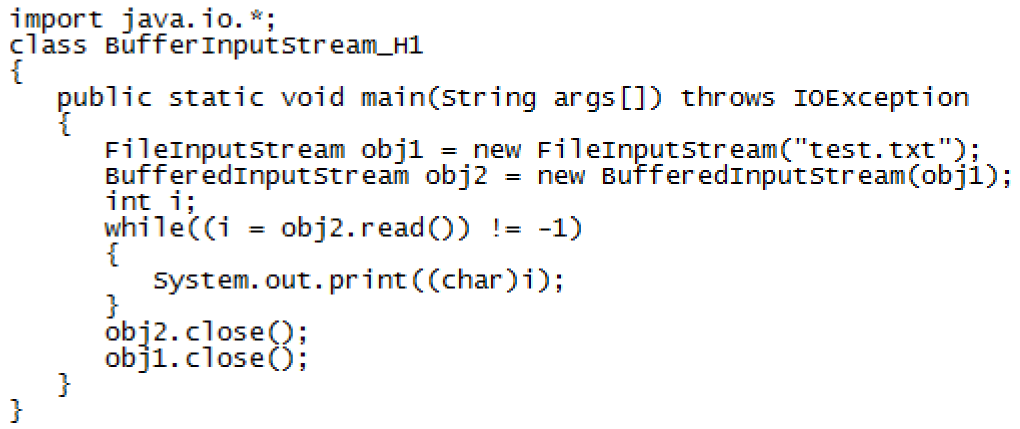
Buffered Byte Streams (Cont...)

- **BufferedInputStream**
  - It allows us to "wrap" any InputStream into a buffered stream and improves performance.
  - There are two **constructors** as mentioned below:
    - BufferedInputStream(InputStream *inputStream*)
    - BufferedInputStream(InputStream *inputStream*, int *bufSize*)

    The 1st form has default buffer size. In the 2nd form, the size of the buffer is passed in bufSize.
  - Buffer size depends on the operating system.
  - A good guess for a size is around 8,192 bytes, and attaching even a small buffer to an I/O stream is always a good idea.

Buffered Byte Streams (Cont...)

- **BufferedInputStream (Cont…)**



```
import java.io.*;
class BufferInputStream_H1
{
   public static void main(String args[]) throws IOException
   {
      FileInputStream obj1 = new FileInputStream("test.txt");
      BufferedInputStream obj2 = new BufferedInputStream(obj1);
      int i;
      while((i = obj2.read()) != -1)
      {
         System.out.print((char)i);
      }
      obj2.close();
      obj1.close();
   }
}
```

Buffered Byte Streams (Cont...)

- **BufferedInputStream (Cont…)**
  - **Output**

```
I want to print this line in my file
```

Buffered Byte Streams (Cont...)

- **BufferedInputStream (Cont…)**

```
import java.io.*;
class BufferInputStream_H
{
  public static void main(String args[]) throws IOException {
    String s = "This is a &copy; copyright symbol" + " but this is &copy not.\n";
    byte buf[] = s.getBytes();
    ByteArrayInputStream in = new ByteArrayInputStream(buf);
    BufferedInputStream f = new BufferedInputStream(in);
    int c;
    boolean marked = false;
    while ((c = f.read()) != -1) {
      switch(c) {
        case '&':
          if (!marked) {
            f.mark(32);
            marked = true;
          } else {
              marked = false;
          }
          break;
        case ';':
          if (marked) {
            marked = false;
            System.out.print("(c)");
          } else
              System.out.print((char)c);
          break;
        case ' ':
          if (marked) {
            marked = false;
            f.reset();
            System.out.print("&");
          } else
              System.out.print((char)c);
          break;
        default:
          if (!marked)
          System.out.print((char)c);
          break;
      }
    }
  }
}
```

Buffered Byte Streams (Cont...)

- **BufferedInputStream (Cont…)**
  - Output

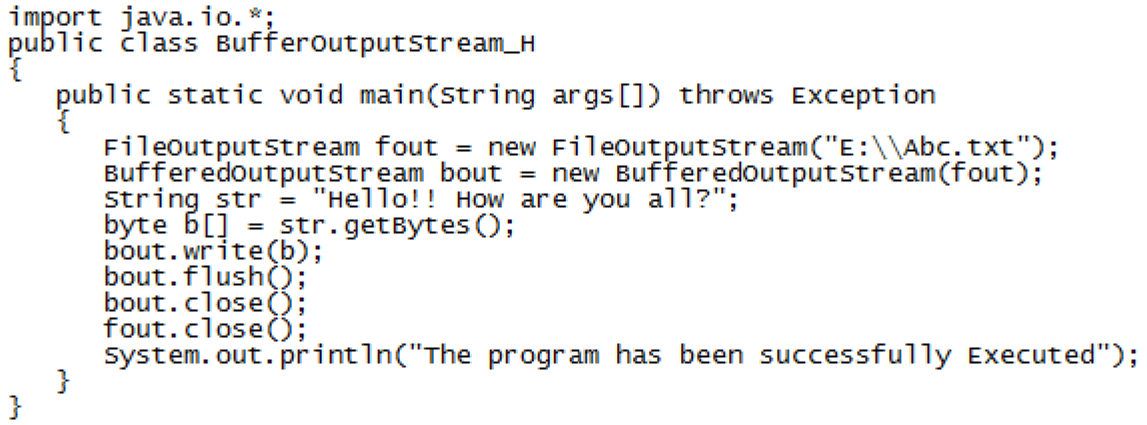
```
This is a (c) copyright symbol but this is &copy not.
```

Buffered Byte Streams (Cont...)

- **BufferedOutputStream**
  - A BufferedOutputStream is similar to any OutputStream with the exception of an added flush( ) method.
  - Flush() method is used to ensure that data buffers are physically written to the actual output device.
  - It internally uses buffer to store data.
  - There are two **constructors**:
    - BufferedOutputStream(OutputStream *outputStream*)
    - BufferedOutputStream(OutputStream *outputStream*, int *bufSize*)

Buffered Byte Streams (Cont...)

- **BufferedOutputStream (Cont…)**



```
import java.io.*;
public class BufferOutputStream_H
{
   public static void main(String args[]) throws Exception
   {
      FileOutputStream fout = new FileOutputStream("E:\\Abc.txt");
      BufferedOutputStream bout = new BufferedOutputStream(fout);
      String str = "Hello!! How are you all?";
      byte b[] = str.getBytes();
      bout.write(b);
      bout.flush();
      bout.close();
      fout.close();
      System.out.println("The program has been successfully Executed");
   }
}
```

Buffered Byte Streams (Cont...)

- **BufferedOutputStream (Cont…)**
  - Output

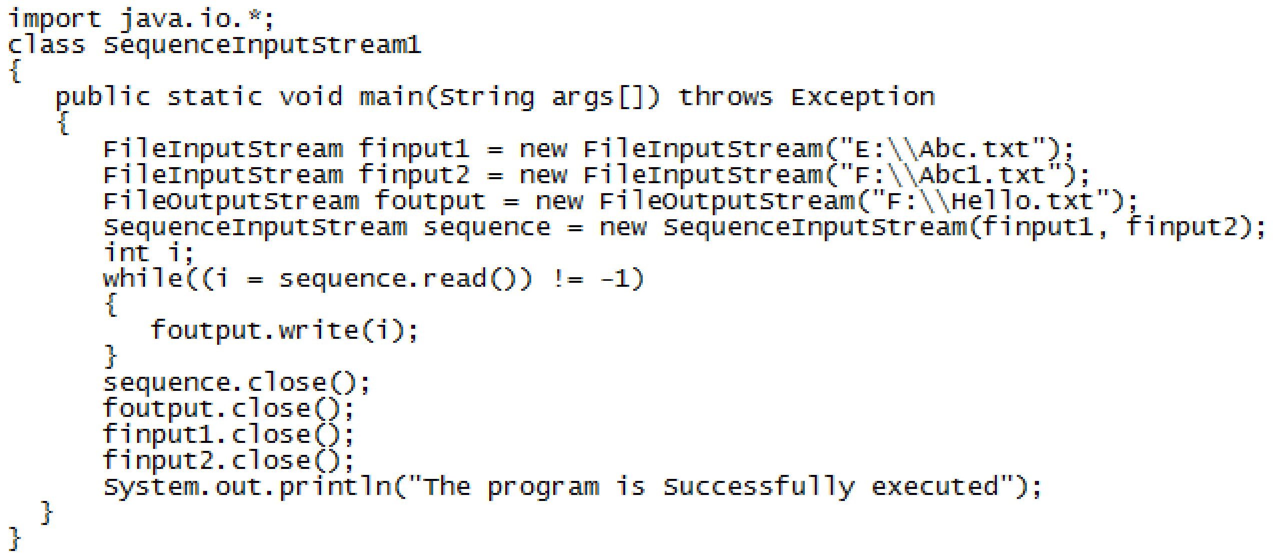
    The program has been successfully Executed

    “Hello!! How are you all?” will be stored in the Abc.txt file.

SequenceInputStream

- The SequenceInputStream class allows us to concatenate on multiple InputStreams.
- Constructor of this class uses either a pair of InputStreams or an Enumeration of InputStreams as its argument:
  - SequenceInputStream(InputStream *first*, InputStream *second)*
  - SequenceInputStream(Enumeration <? extends InputStream> *streamEnum)*
- If we need to read the data from more than two files, we need to use Enumeration. Enumeration object can be obtained by calling elements() method of the Vector class.

SequenceInputStream (Cont…)



```
import java.io.*;
class SequenceInputStream1
{
   public static void main(String args[]) throws Exception
   {
      FileInputStream finput1 = new FileInputStream("E:\\Abc.txt");
      FileInputStream finput2 = new FileInputStream("F:\\Abc1.txt");
      FileOutputStream foutput = new FileOutputStream("F:\\Hello.txt");
      SequenceInputStream sequence = new SequenceInputStream(finput1, finput2);
      int i;
      while((i = sequence.read()) != -1)
      {
         foutput.write(i);
      }
      sequence.close();
      foutput.close();
      finput1.close();
      finput2.close();
      System.out.println("The program is Successfully executed");
  }
}
```

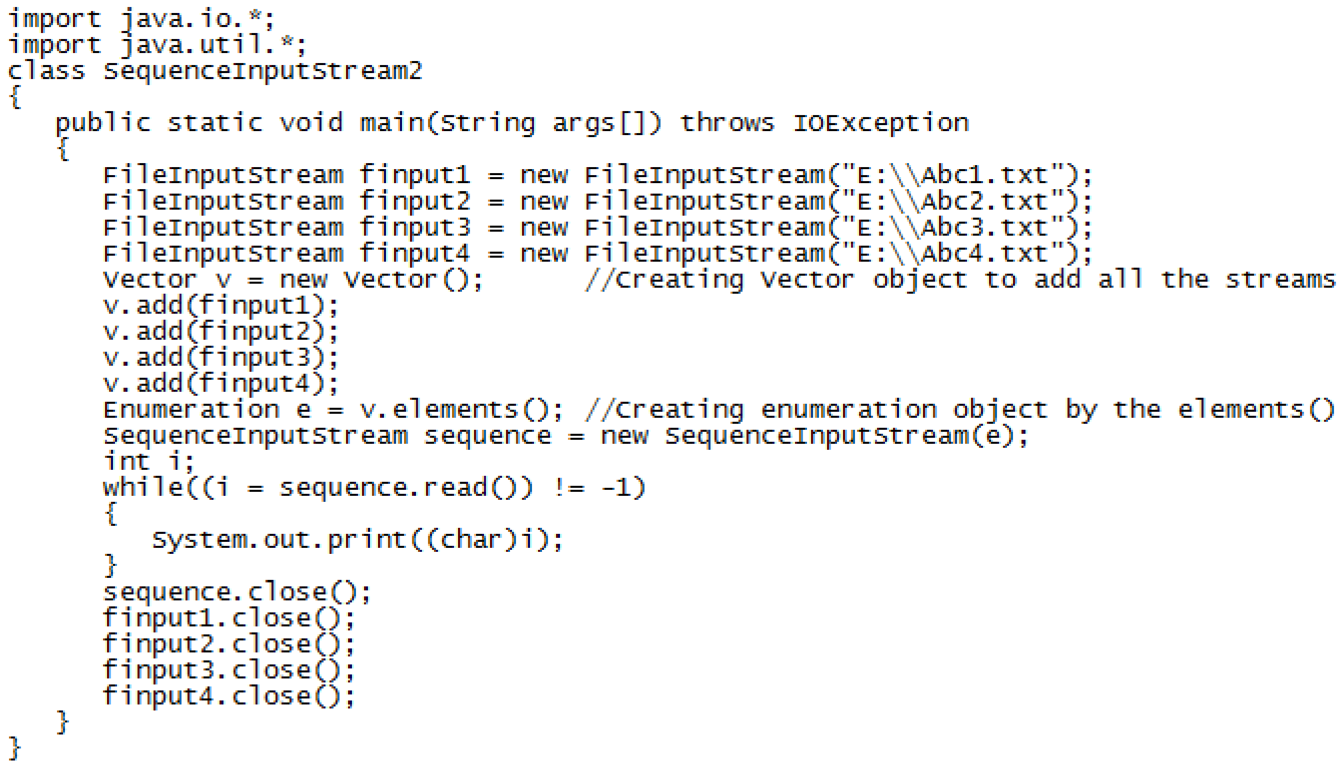
SequenceInputStream (Cont...)

- **Output**

  The program is Successfully executed

  Content of both the files will be written in Hello.txt file.

SequenceInputStream (Cont...)



```
import java.io.*;
import java.util.*;
class SequenceInputStream2
{
   public static void main(String args[]) throws IOException
   {
      FileInputStream finput1 = new FileInputStream("E:\\Abc1.txt");
      FileInputStream finput2 = new FileInputStream("E:\\Abc2.txt");
      FileInputStream finput3 = new FileInputStream("E:\\Abc3.txt");
      FileInputStream finput4 = new FileInputStream("E:\\Abc4.txt");
      Vector v = new Vector();      //Creating Vector object to add all the streams
      v.add(finput1);
      v.add(finput2);
      v.add(finput3);
      v.add(finput4);
      Enumeration e = v.elements(); //Creating enumeration object by the elements()
      SequenceInputStream sequence = new SequenceInputStream(e);
      int i;
      while((i = sequence.read()) != -1)
      {
         System.out.print((char)i);
      }
      sequence.close();
      finput1.close();
      finput2.close();
      finput3.close();
      finput4.close();
   }
}
```

SequenceInputStream (Cont...)

- **Output**

```
Hello!! How are you?
```

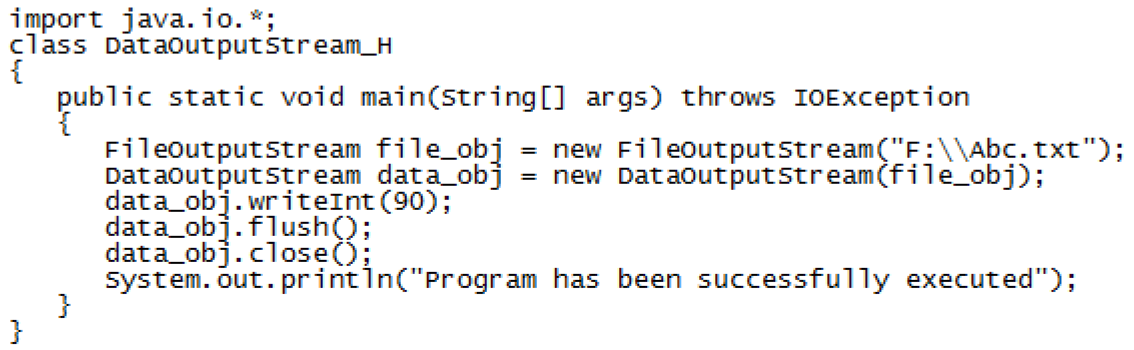
Content of different files will be shown on the console

DataOutputStream

- This class allows an application to write primitive data types to the output stream.
- It implements DataOutput interface.
- DataOutput defines methods to convert values of a primitive type into a byte sequence. Then, writes it to the stream.
- DataOutputStream extends FilterOutputStream, which extends OutputStream.
- It has one constructor:

  DataOutputStream(OutputStream *outputStream*) //*outputStream* specifies the output stream to which data will be written

DataOutputStream (Cont...)



```
import java.io.*;
class DataOutputStream_H
{
   public static void main(String[] args) throws IOException
   {
      FileOutputStream file_obj = new FileOutputStream("F:\\Abc.txt");
      DataOutputStream data_obj = new DataOutputStream(file_obj);
      data_obj.writeInt(90);
      data_obj.flush();
      data_obj.close();
      System.out.println("Program has been successfully executed");
   }
}
```

DataOutputStream (Cont...)

- **Output**

  Program has been successfully executed

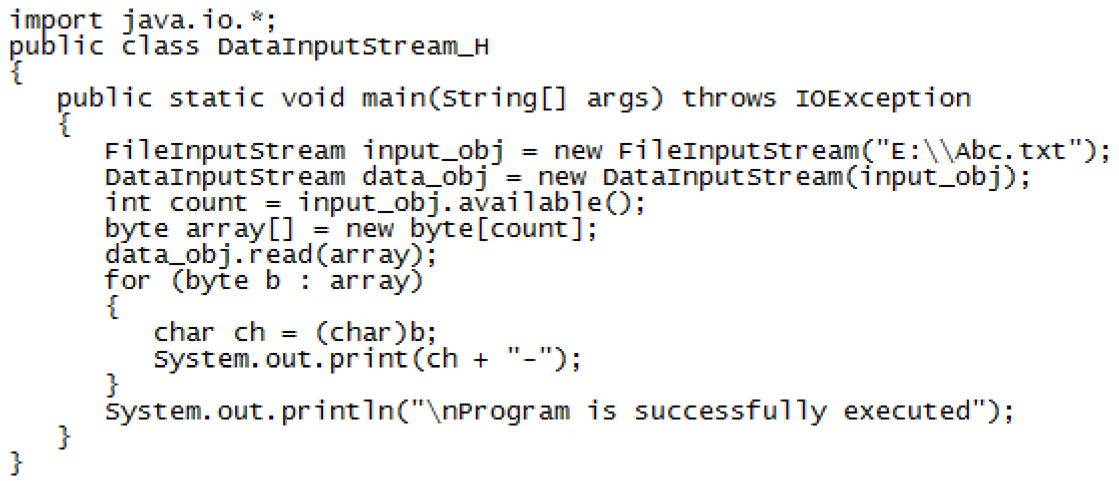
  In the file, “Z” will be stored.

DataInputStream

- DataInputStream class allows an application to read primitive data from the input stream.
- This class implements DataIntput interface.
- DataInput defines methods to read byte sequence. Then, converts into a primitive type.
- DataInputStream extends FilterInputStream, which extends InputStream.
- It has one **constructor**, which is mentioned below:

DataInputStream(InputStream *InputStream)*

DataInputStream (Cont...)



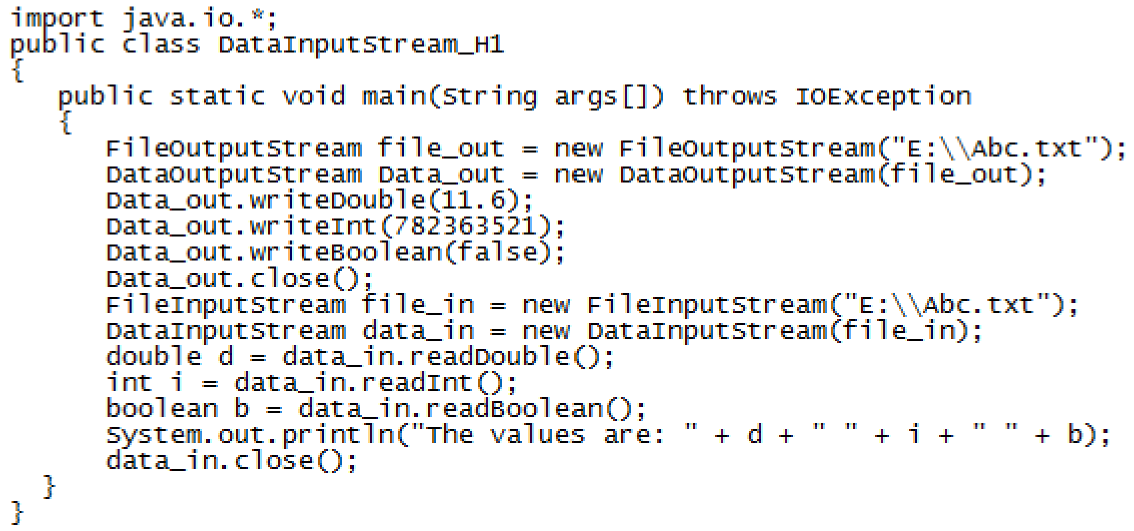
```
import java.io.*;
public class DataInputStream_H
{
   public static void main(String[] args) throws IOException
   {
      FileInputStream input_obj = new FileInputStream("E:\\Abc.txt");
      DataInputStream data_obj = new DataInputStream(input_obj);
      int count = input_obj.available();
      byte array[] = new byte[count];
      data_obj.read(array);
      for (byte b : array)
      {
         char ch = (char)b;
         System.out.print(ch + "-");
      }
      System.out.println("\nProgram is successfully executed");
   }
}
```

DataInputStream (Cont...)

- **Output**

```
H-e-l-l-o-
Program is successfully executed
```

Combination of Both DataOutputStream and DataInputStream



```
import java.io.*;
public class DataInputStream_H1
{
   public static void main(String args[]) throws IOException
   {
      FileOutputStream file_out = new FileOutputStream("E:\\Abc.txt");
      DataOutputStream Data_out = new DataOutputStream(file_out);
      Data_out.writeDouble(11.6);
      Data_out.writeInt(782363521);
      Data_out.writeBoolean(false);
      Data_out.close();
      FileInputStream file_in = new FileInputStream("E:\\Abc.txt");
      DataInputStream data_in = new DataInputStream(file_in);
      double d = data_in.readDouble();
      int i = data_in.readInt();
      boolean b = data_in.readBoolean();
      System.out.println("The values are: " + d + " " + i + " " + b);
      data_in.close();
   }
}
```

Combination of Both DataOutputStream and DataInputStream (Cont...)

- **Output**

```
The values are: 11.6 782363521 false
```

RandomAccessFile

- This class is used for reading and writing to random access file.
- It implements DataInput and DataOutput interfaces that define basic I/O methods.
- In addition, it implements Closeable interface.
- This class is special as it supports positioning requests like cursor. We can position the file pointer within the file.
- If end-of-file is reached before the desired number of byte has been read, then EOFException is thrown (a type of IOException).

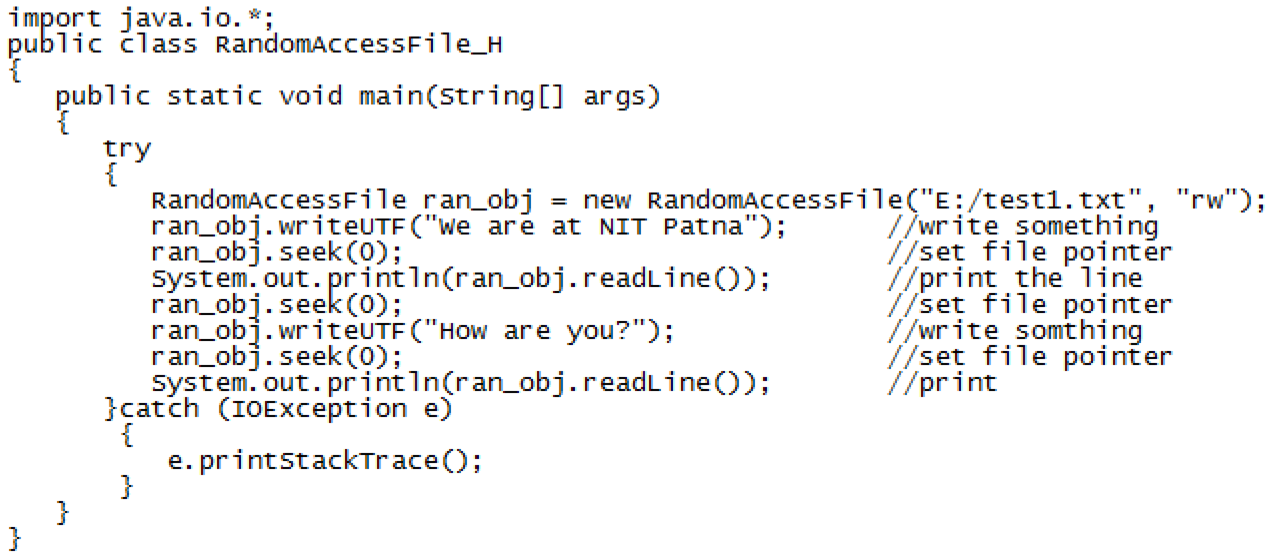
RandomAccessFile (Cont…)

- There are two **constructors** of this class:
  - RandomAccessFile(File *fileObj*, String *access*) throws FileNotFoundException
  - RandomAccessFile(String *filename*, String *access*) throws FileNotFoundException

  In the 1st form, *fileObj* specifies the name of the file to open as a File object. In the 2nd form, the name of the file is passed in filename.

  In both the cases, *access* determines what type of file accesses i.e. “r”, “rw”, etc. are permitted.
- seek( ) is used to set the current position of the file pointer within the file.

RandomAccessFile (Cont...)



```
import java.io.*;
public class RandomAccessFile_H
{
   public static void main(String[] args)
   {
      try
      {
         RandomAccessFile ran_obj = new RandomAccessFile("E:/test1.txt", "rw");
         ran_obj.writeUTF("We are at NIT Patna");      //write something
         ran_obj.seek(0);                              //set file pointer
         System.out.println(ran_obj.readLine());       //print the line
         ran_obj.seek(0);                              //set file pointer
         ran_obj.writeUTF("How are you?");             //write somthing
         ran_obj.seek(0);                              //set file pointer
         System.out.println(ran_obj.readLine());       //print
      }catch (IOException e)
       {
          e.printStackTrace();
       }
   }
}
```

RandomAccessFile (Cont...)

- **Output**

  In the File “How are you?T Patna” will be stored.

  In the console: We are at NIT Patna

  ```
  We are at NIT Patna
  How are you?T Patna
  ```

Character Streams

- Character stream is made for character.
- It supports "write once, run anywhere" property.
- At the top, there are two abstract classes:
  - **Reader:** Reader defines streaming character input. Most of the classes throw IOException. It implements the Closeable and Readable interfaces.
  - **Writer:** Writer defines streaming character output. It implements the Closeable, Flushable and Appendable interfaces. All the methods of this class throw an IOException.

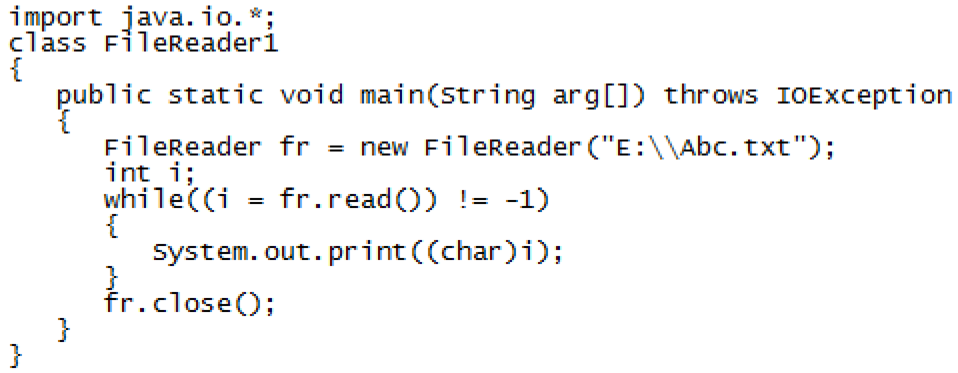
FileReader

- The FileReader class creates a Reader that we can use to read the contents of a file.

- There are two **constructors** as mentioned below:
  - FileReader(String *filePath*) //*filepath* is the path of a file
  - FileReader(File *fileObj*) //*fileObj* is the file object that depict the file

- We can also throw a FileNotFoundException.

FileReader (Cont...)



```
import java.io.*;
class FileReader1
{
   public static void main(String arg[]) throws IOException
   {
      FileReader fr = new FileReader("E:\\Abc.txt");
      int i;
      while((i = fr.read()) != -1)
      {
         System.out.print((char)i);
      }
      fr.close();
   }
}
```

FileReader (Cont...)

- **Output**

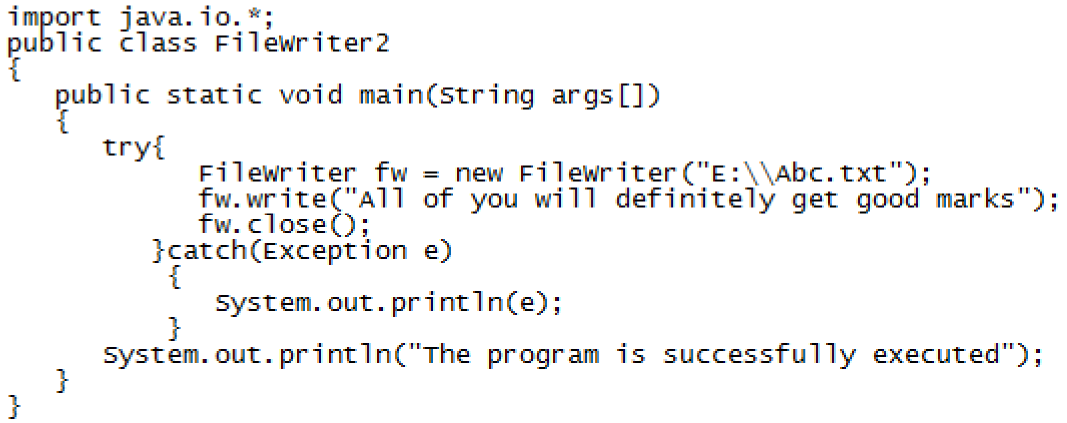
```
Hello Dear!!
Give me a party.
```

FileWriter

- FileWriter class creates a Writer that we can use to write to a file.
- Unlike FileOutputStream, we don't need to convert string into byte array because it provides method to write string directly.
- There are four **constructors**:
  - FileWriter(String *filePath*)
  - FileWriter(String *filePath*, boolean *append*)
  - FileWriter(File *fileObj*)
  - FileWriter(File *fileObj*, boolean *append*)

  In the above constructors, if append is true, we can append the output at the end of the file.

FileWriter (Cont...)



```
import java.io.*;
public class FileWriter2
{
   public static void main(String args[])
   {
      try{
            FileWriter fw = new FileWriter("E:\\Abc.txt");
            fw.write("All of you will definitely get good marks");
            fw.close();
         }catch(Exception e)
          {
             System.out.println(e);
          }
      System.out.println("The program is successfully executed");
   }
}
```

FileWriter (Cont...)

- **Output**

  The program is successfully executed

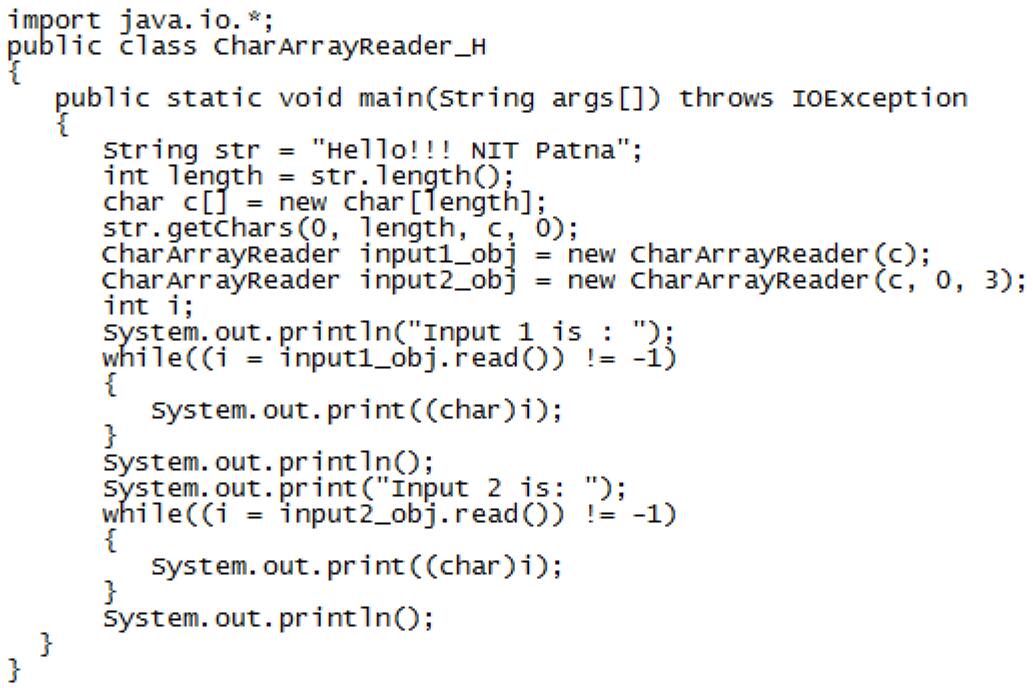
  In the file, “All of you will definitely get good marks” will be stored.

CharArrayReader

- CharArrayReader is an implementation of an input stream that uses a character array as the source (This class implements a character buffer that can be used as a character-input stream).
- It inherits Reader class.
- There are two **constructors** as mentioned below:
  - CharArrayReader(char *array[ ]*)
  - CharArrayReader(char *array[ ]*, int *start*, int *numChars*)

  Here, array is the input source. The 2nd constructor creates a Reader from a subset of our character array that begins with the character at the index specified by start and is numChars long.

CharArrayReader (Cont...)



```java
import java.io.*;
public class CharArrayReader_H
{
   public static void main(String args[]) throws IOException
   {
      String str = "Hello!!! NIT Patna";
      int length = str.length();
      char c[] = new char[length];
      str.getChars(0, length, c, 0);
      CharArrayReader input1_obj = new CharArrayReader(c);
      CharArrayReader input2_obj = new CharArrayReader(c, 0, 3);
      int i;
      System.out.println("Input 1 is : ");
      while((i = input1_obj.read()) != -1)
      {
         System.out.print((char)i);
      }
      System.out.println();
      System.out.print("Input 2 is: ");
      while((i = input2_obj.read()) != -1)
      {
         System.out.print((char)i);
      }
      System.out.println();
  }
}
```

CharArrayReader (Cont...)

- **Output**

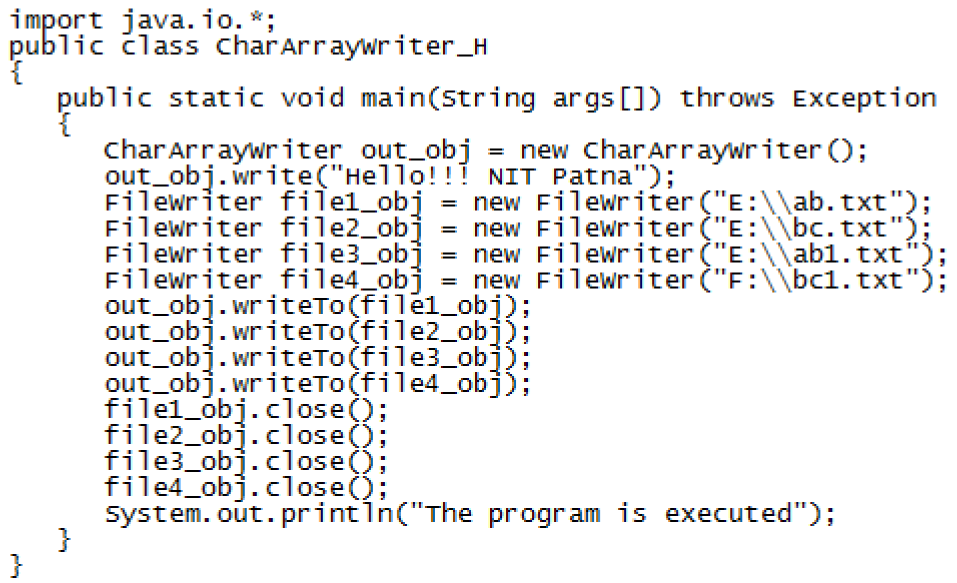
```
Input 1 is :
Hello!!!  NIT Patna
Input 2 is: Hel
```

CharArrayWriter

- This class implements a character buffer that can be used as an Writer. The buffer automatically grows, when data is written to the stream.
- The CharArrayWriter class can be used to write common data to multiple files. This class inherits Writer class.
- The data can be retrieved using toCharArray() and toString().
- There are two **constructors**:
  - CharArrayWriter( )
  - CharArrayWriter(int *numChars*)

  In the 1st form, a buffer with a default size is created. In the 2nd form, a buffer is created with a size equal to that specified by numChars. The buffer size will be increased automatically.

CharArrayWriter (Cont...)



```
import java.io.*;
public class CharArrayWriter_H
{
   public static void main(String args[]) throws Exception
   {
      CharArrayWriter out_obj = new CharArrayWriter();
      out_obj.write("Hello!!! NIT Patna");
      FileWriter file1_obj = new FileWriter("E:\\ab.txt");
      FileWriter file2_obj = new FileWriter("E:\\bc.txt");
      FileWriter file3_obj = new FileWriter("E:\\ab1.txt");
      FileWriter file4_obj = new FileWriter("F:\\bc1.txt");
      out_obj.writeTo(file1_obj);
      out_obj.writeTo(file2_obj);
      out_obj.writeTo(file3_obj);
      out_obj.writeTo(file4_obj);
      file1_obj.close();
      file2_obj.close();
      file3_obj.close();
      file4_obj.close();
      System.out.println("The program is executed");
   }
}
```

CharArrayWriter (Cont...)

- **Output**

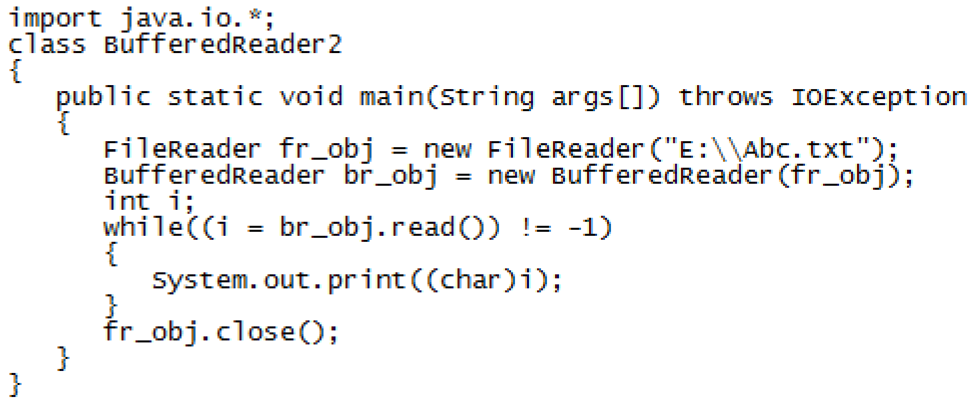
  The program is executed

  In all the files, “Hello!!! NIT Patna” will be written.

BufferedReader

- Java BufferedReader class is used to read the text from a character based input stream.
- It can be used to read data line by line by readLine() method.
- It makes the performance fast.
- There are two **constructors** of this class:
  - BufferedReader(Reader *inputStream*)
  - BufferedReader(Reader *inputStream*, int *bufSize*)
- To support moving backward in the stream within the available buffer, it implements the mark( ) and reset( ).

BufferedReader (Cont...)



```
import java.io.*;
class BufferedReader2
{
   public static void main(String args[]) throws IOException
   {
      FileReader fr_obj = new FileReader("E:\\Abc.txt");
      BufferedReader br_obj = new BufferedReader(fr_obj);
      int i;
      while((i = br_obj.read()) != -1)
      {
         System.out.print((char)i);
      }
      fr_obj.close();
   }
}
```

BufferedReader (Cont...)

- **Output**

```
How are you all?
```

BufferedReader (Cont...)

```java
import java.io.*;
class BufferedReader1
{
  public static void main(String args[]) throws IOException
  {
    String s = "This is a &copy; copyright symbol " + "but this is &copy not.\n";
    char buf[] = new char[s.length()];
    s.getChars(0, s.length(), buf, 0);
    CharArrayReader in = new CharArrayReader(buf);
    BufferedReader f = new BufferedReader(in);
    int c;
    boolean marked = false;
    while ((c = f.read()) != -1) {
      switch(c) {
        case '&':
          if (!marked) {
            f.mark(32);
            marked = true;
          } else{
              marked = false;
            }
          break;
        case ';':
          if (marked) {
            marked = false;
            System.out.print("(c)");
          } else
             System.out.print((char) c);
          break;
        case ' ':
          if (marked) {
            marked = false;
            f.reset();
            System.out.print("&");
          } else
              System.out.print((char) c);
          break;
        default:
          if (!marked)
             System.out.print((char) c);
          break;
      }
    }
  }
}
```

BufferedReader (Cont...)

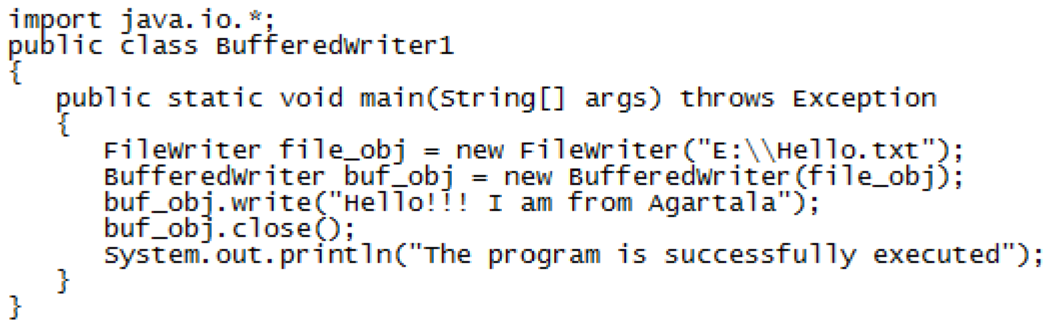
- **Output**

```
This is a (c) copyright symbol but this is &copy not.
```

BufferedWriter

- BufferedWriter class is used to offer buffering for Writer instances.
- It inherits Writer class.
- Using a BufferedWriter can increase performance by reducing the number of times data is actually physically written to the output stream.
- There are two **constructors**:
  - BufferedWriter(Writer *outputStream)*
  - BufferedWriter(Writer *outputStream*, int *bufSize)*

BufferedWriter (Cont...)



```
import java.io.*;
public class BufferedWriter1
{
   public static void main(String[] args) throws Exception
   {
      FileWriter file_obj = new FileWriter("E:\\Hello.txt");
      BufferedWriter buf_obj = new BufferedWriter(file_obj);
      buf_obj.write("Hello!!! I am from Agartala");
      buf_obj.close();
      System.out.println("The program is successfully executed");
   }
}
```

BufferedWriter (Cont...)

- **Output**

The program is successfully executed

In the file “Hello!!! I am from Agartala” will be saved.

PushbackReader

- The PushbackReader class allows one or more characters to be returned to the input stream.
- It allows us to look ahead in the input stream.
- There are two **constructors**:
  - PushbackReader(Reader *inputStream*)
  - PushbackReader(Reader *inputStream*, int *bufSize*)

  The 1st form creates a buffered stream that allows one character to be pushed back. In the 2nd form, the size of the pushback buffer is passed in bufSize.

PushbackReader (Cont…)

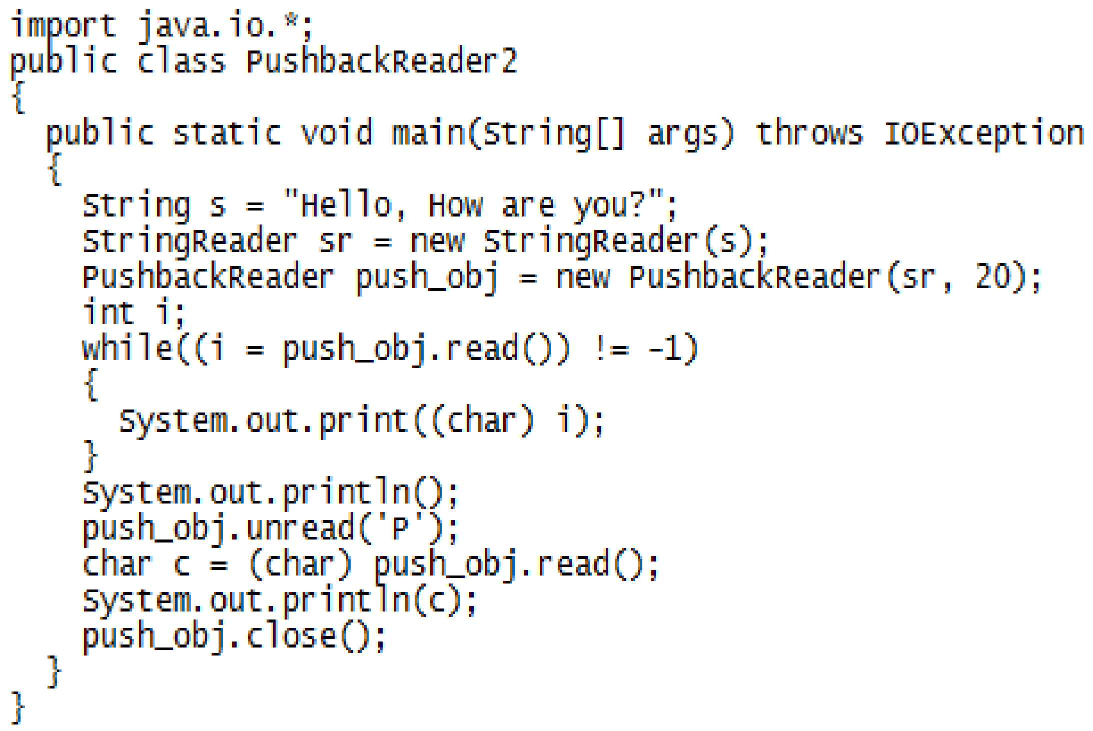
- This class provides unread( ), which returns one or more characters to the invoking input stream. It has three forms:
  - void unread(int *ch*)
  - void unread(char *buffer[ ]*)
  - void unread(char *buffer[ ]*, int *offset*, int *numChars*)

  The 1st form pushes back the character passed in ch. It will be the next character returned by a subsequent call to read( ).

  The 2nd form returns the characters in buffer.

  The 3rd form pushes back numChars characters beginning at offset from buffer.

PushbackReader (Cont...)



```
import java.io.*;
public class PushbackReader2
{
  public static void main(String[] args) throws IOException
  {
    String s = "Hello, How are you?";
    StringReader sr = new StringReader(s);
    PushbackReader push_obj = new PushbackReader(sr, 20);
    int i;
    while((i = push_obj.read()) != -1)
    {
      System.out.print((char) i);
    }
    System.out.println();
    push_obj.unread('P');
    char c = (char) push_obj.read();
    System.out.println(c);
    push_obj.close();
  }
}
```

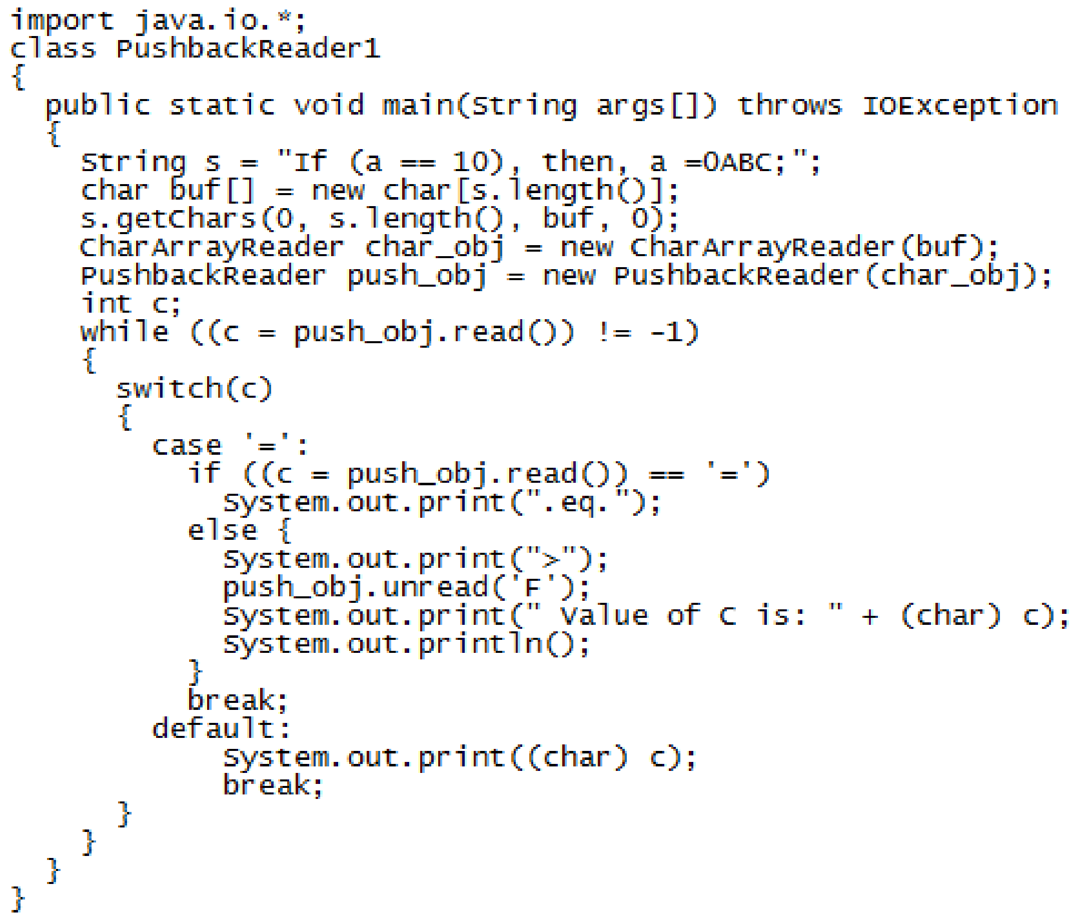
PushbackReader (Cont...)

- **Output**

```
Hello, How are you?

P
```

PushbackReader (Cont...)



```
import java.io.*;
class PushbackReader1
{
  public static void main(String args[]) throws IOException
  {
    String s = "If (a == 10), then, a =0ABC;";
    char buf[] = new char[s.length()];
    s.getChars(0, s.length(), buf, 0);
    CharArrayReader char_obj = new CharArrayReader(buf);
    PushbackReader push_obj = new PushbackReader(char_obj);
    int c;
    while ((c = push_obj.read()) != -1)
    {
      switch(c)
      {
        case '=':
          if ((c = push_obj.read()) == '=')
            System.out.print(".eq.");
          else {
            System.out.print(">");
            push_obj.unread('F');
            System.out.print(" Value of C is: " + (char) c);
            System.out.println();
          }
          break;
        default:
            System.out.print((char) c);
            break;
      }
    }
  }
}
```

PushbackReader (Cont...)

- **Output**

```
If (a .eq. 10), then, a > Value of C is: 0

FABC;
```

PrintWriter

- PrintWriter is essentially a character-oriented version of PrintStream.
- This class is the implementation of Writer class.
- It is used to print the formatted representation of objects to the text-output stream.
- This class converts the primitive data (int, float, etc.) into the text format. Then, writes that formatted data to the writer.
- It is useful when we generate an output, where we have to mix text and numbers.

PrintWriter (Cont…)

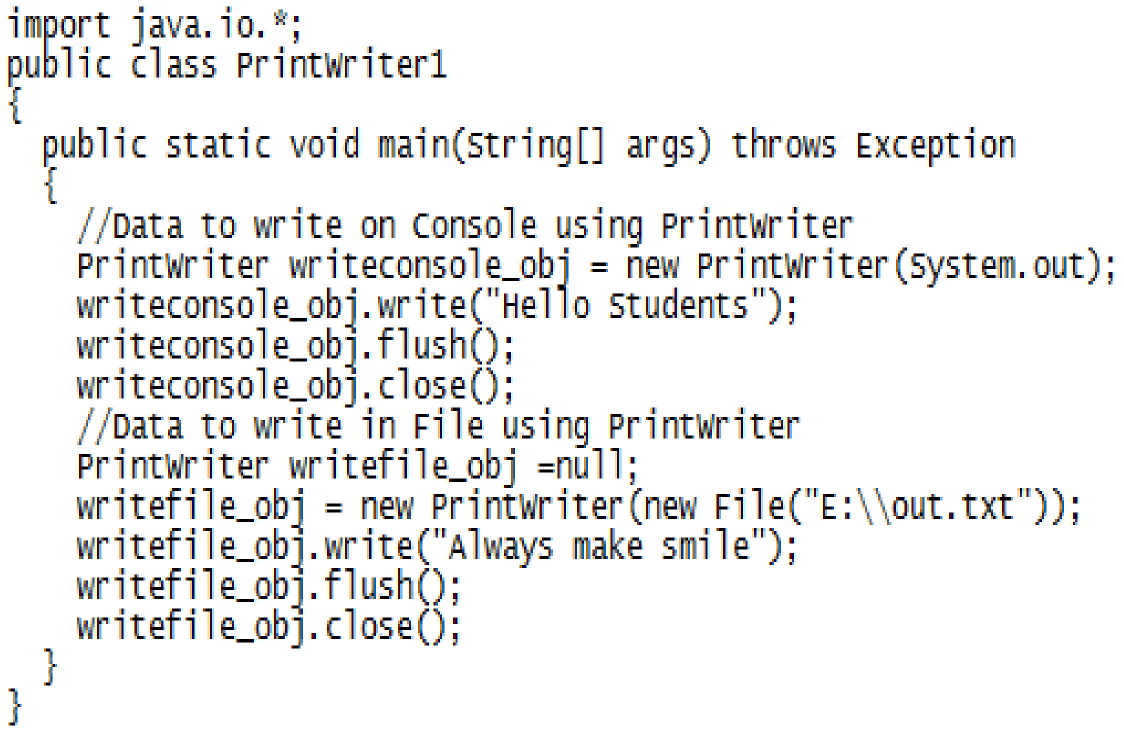
- There are many **constructors**. Some are mentioned below:
  - PrintWriter(OutputStream *outputStream*)
  - PrintWriter(OutputStream *outputStream*, boolean *flushOnNewline*)
  - PrintWriter(Writer *outputStream*)
  - PrintWriter(Writer *outputStream*, boolean *flushOnNewline*)

  Here, outputStream specifies an open OutputStream that will receive output.

  The flushOnNewline parameter controls whether the output buffer is automatically flushed every time or not. If it is true, flushing automatically takes place.

  Constructors that do not specify the flushOnNewline parameter, do not automatically flush.

PrintWriter (Cont...)



```
import java.io.*;
public class PrintWriter1
{
  public static void main(String[] args) throws Exception
  {
    //Data to write on Console using PrintWriter
    PrintWriter writeconsole_obj = new PrintWriter(System.out);
    writeconsole_obj.write("Hello Students");
    writeconsole_obj.flush();
    writeconsole_obj.close();
    //Data to write in File using PrintWriter
    PrintWriter writefile_obj =null;
    writefile_obj = new PrintWriter(new File("E:\\out.txt"));
    writefile_obj.write("Always make smile");
    writefile_obj.flush();
    writefile_obj.close();
  }
}
```

PrintWriter (Cont...)

- **Output**

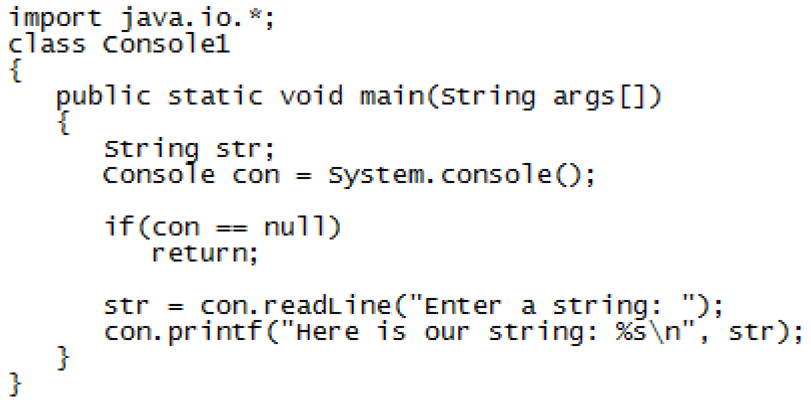
```
Hello Students
```

In the output file “Always make smile” will be saved.

Console Class

- Console class is used to read from and write to the console, if one exists.
- It implements the Flushable interface.
- Console is convenience because its functionality is available through System.in and System.out.
- Console does not have constructors. Instead, a Console object is obtained by calling System.console( ).
- If a console is available, then a reference to it is returned. Otherwise, null is returned.

Console Class (Cont...)



```java
import java.io.*;
class Console1
{
   public static void main(String args[])
   {
      String str;
      Console con = System.console();

      if(con == null)
         return;

      str = con.readLine("Enter a string: ");
      con.printf("Here is our string: %s\n", str);
   }
}
```

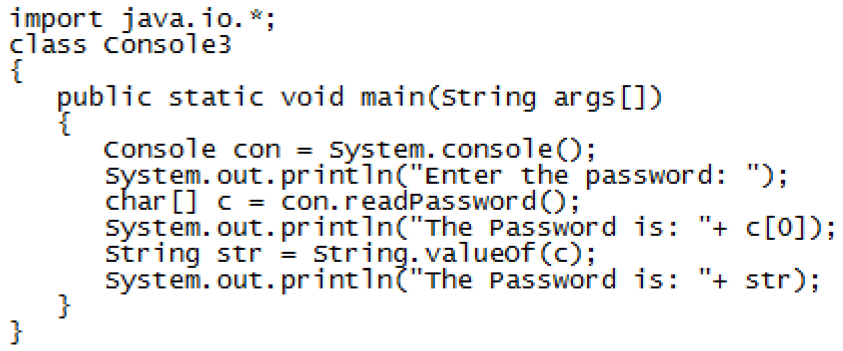
Console Class (Cont...)

- **Output**

```
Enter a string:
Here is our string:
```

Based on the input, output would be shown

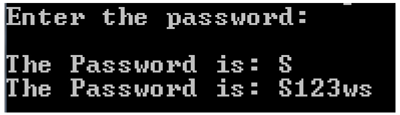
Console Class (Cont...)



```
import java.io.*;
class Console3
{
   public static void main(String args[])
   {
      Console con = System.console();
      System.out.println("Enter the password: ");
      char[] c = con.readPassword();
      System.out.println("The Password is: "+ c[0]);
      String str = String.valueOf(c);
      System.out.println("The Password is: "+ str);
   }
}
```

Console Class (Cont...)

- **Output**



```
Enter the password:

The Password is: S
The Password is: S123ws
```

Using Stream IO

- The wc( ) method operates on any input stream and counts the number of characters, lines and words.
- When executed with no arguments, WordCount creates an InputStreamReader object using System.in as the source for the stream. This stream is then passed to wc( ).
- When executed with one or more arguments, WordCount assumes that these are filenames and creates FileReaders for each of them. Then, the resultant FileReader objects are passed to the wc( ) method.

Using Stream IO (Cont...)

```
import java.io.*;
class WordCount1
{
  public static int words = 0;
  public static int lines = 0;
  public static int chars = 0;
  public static void wc(InputStreamReader isr)
  throws IOException {
    int c = 0;
    boolean lastWhite = true;
    String whiteSpace = " \t\n\r";
    while ((c = isr.read()) != -1)
    {
      chars++;                              //Count characters
      if (c == '\n') {                      //Count lines
        lines++;
      }
      int index = whiteSpace.indexOf(c); //Count words by detecting the start of a word
      if(index == -1) {                     //If space found, return 0
        if(lastWhite == true) {
          ++words;
        }
        lastWhite = false;
      }
      else {
        lastWhite = true;
      }
    }
    if(chars != 0) {
      ++lines;
    }
  }
  public static void main(String args[])
  {
    FileReader fr;
    try {
      if (args.length == 0) {              //We are working with Standard Input
        wc(new InputStreamReader(System.in));
      }
      else {                                //We are paasing a list of files
        for (int i = 0; i < args.length; i++) {
          fr = new FileReader(args[i]);
          wc(fr);
        }
      }
    } catch (IOException e) {
        return;
      }
    System.out.println(lines + " " + words + " " + chars);
  }
}
```

Using Stream IO (Cont...)

- **Output**

  After compiling the program, we have to give file name in the command line, such as java class_name *file_name .txt*

Serialization

- **Serialization** is the process of writing the state of an object to a byte stream.
- The reverse operation is called **Deserialization**, where byte-stream is converted into an object.
- This is useful, when we want to save the state of the program to a persistent storage media.
- It supports to implement Remote Method Invocation (RMI). RMI allows a Java object on one machine to invoke a method of a Java object on a different machine.
- If we do not want any field to be a part of the object state, we declare it either **static or transient**.

Interfaces and Classes of Serialization

- **Serializable**
  - A class must implement **Serializable** interface presents in java.io package in order to serialize its object successfully.
  - It is simply used to indicate that a class may be serialized (marker interface).
  - If a class is serializable, all of its subclasses are also serializable.
- **Externalizable**
  - It is used to control the save and restore (state of an object) process.
  - There are two methods defined by Externalizable interface:
    - void readExternal(ObjectInput *inStream*) throws IOException, ClassNotFoundException
    - void writeExternal(ObjectOutput *outStream*) throws IOException

    inStream is the byte stream from which the object is to be read, and outStream is the byte stream to which the object is to be written.

Interfaces and Classes of Serialization (Cont...)

- **ObjectOutput**
  - This interface extends the DataOutput interface and supports object serialization. There are many methods in ObjectOutput interface.
  - **writeObject( ) method is called to serialize an object.**
  - All the methods throw an IOException on error conditions.
- **ObjectOutputStream**
  - The ObjectOutputStream class extends the OutputStream class and implements the ObjectOutput interface.
  - It is responsible for writing objects to a stream.
  - There is a constructor of this class:

    ObjectOutputStream(OutputStream *outStream)* throws IOException

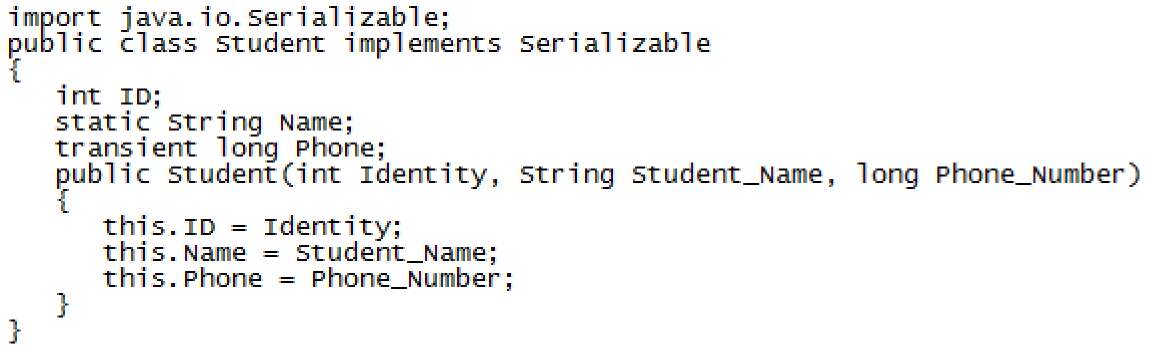
Interfaces and Classes of Serialization (Cont…)

- **ObjectInput**
  - This interface extends the DataInput interface. It supports object serialization.
  - **readObject( ) method is called to deserialize an object.**
  - This interface has many methods, which throw an IOException, and readObject() method can also throw ClassNotFoundException.
- **ObjectInputStream**
  - This class extends the InputStream class and implements the ObjectInput interface. ObjectInputStream is responsible for reading objects from a stream.
  - There is a constructor of this class:

    ObjectInputStream(InputStream *inStream*) throws IOException

Example of Serialization

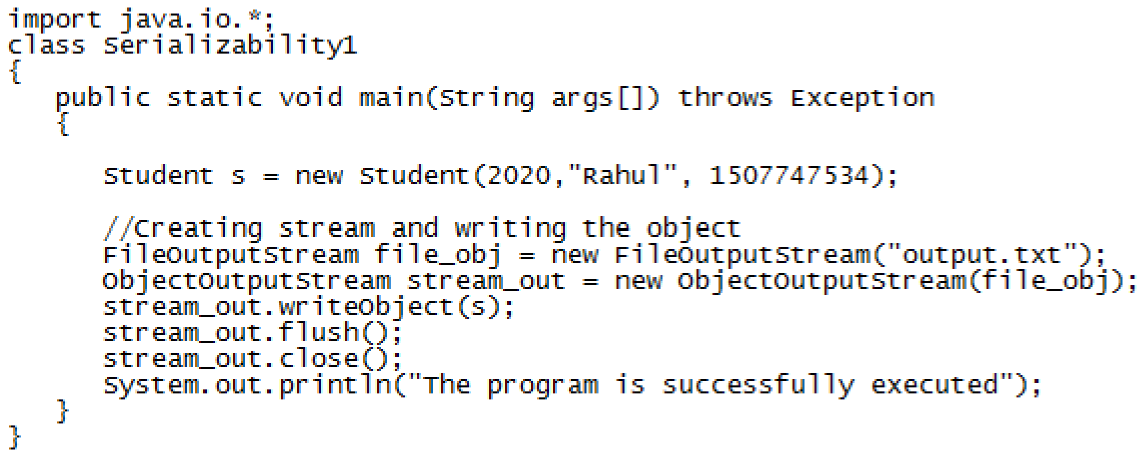
- Creating a class that has Serializable property



```
import java.io.Serializable;
public class Student implements Serializable
{
   int ID;
   static String Name;
   transient long Phone;
   public Student(int Identity, String Student_Name, long Phone_Number)
   {
      this.ID = Identity;
      this.Name = Student_Name;
      this.Phone = Phone_Number;
   }
}
```

Example of Serialization (Cont..)

- Serializing the object of Student class (Last Slide)



```
import java.io.*;
class Serializability1
{
   public static void main(String args[]) throws Exception
   {

      Student s = new Student(2020,"Rahul", 1507747534);

      //Creating stream and writing the object
      FileOutputStream file_obj = new FileOutputStream("output.txt");
      ObjectOutputStream stream_out = new ObjectOutputStream(file_obj);
      stream_out.writeObject(s);
      stream_out.flush();
      stream_out.close();
      System.out.println("The program is successfully executed");
   }
}
```

Example of Serialization (Cont..)

- **Run Serializable1 class**

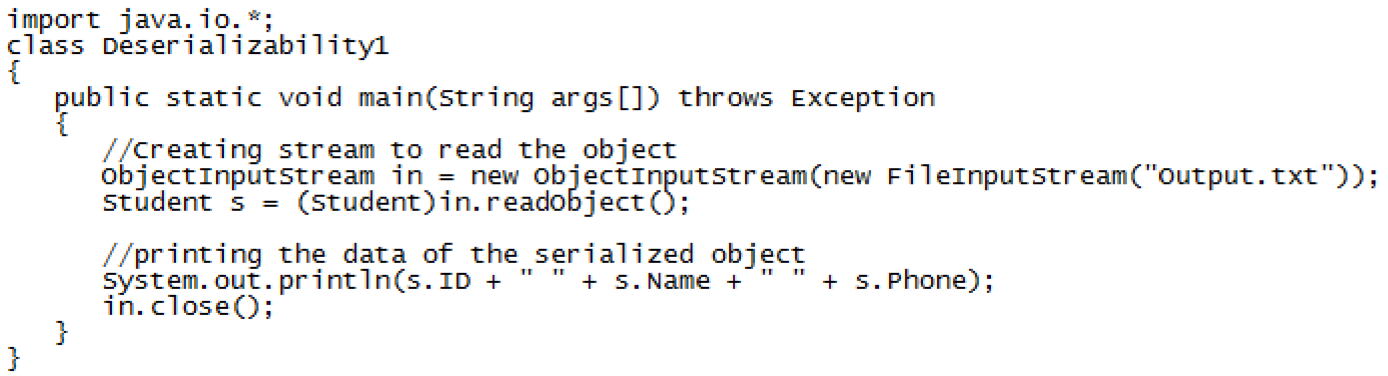
- **Output**

  The program is successfully executed

  A new file “output.txt” will be generated. In this file, the state of the object will be stored.

Example of Serialization (Cont..)

- Deserialization process to get the data



```
import java.io.*;
class Deserializability1
{
   public static void main(String args[]) throws Exception
   {
      //Creating stream to read the object
      ObjectInputStream in = new ObjectInputStream(new FileInputStream("Output.txt"));
      Student s = (Student)in.readObject();

      //printing the data of the serialized object
      System.out.println(s.ID + " " + s.Name + " " + s.Phone);
      in.close();
   }
}
```

Example of Serialization (Cont..)

- **Run Deserializable1 class**

- **Output**

```
2020  Null  0
```

thank
you . . .

Slides are prepared from various sources, such as Book, Internet Links and many more.