OBJECT ORIENTED PROGRAMMING WITH JAVA

Input-Output Handling in Java – I

Debasis Samanta
Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

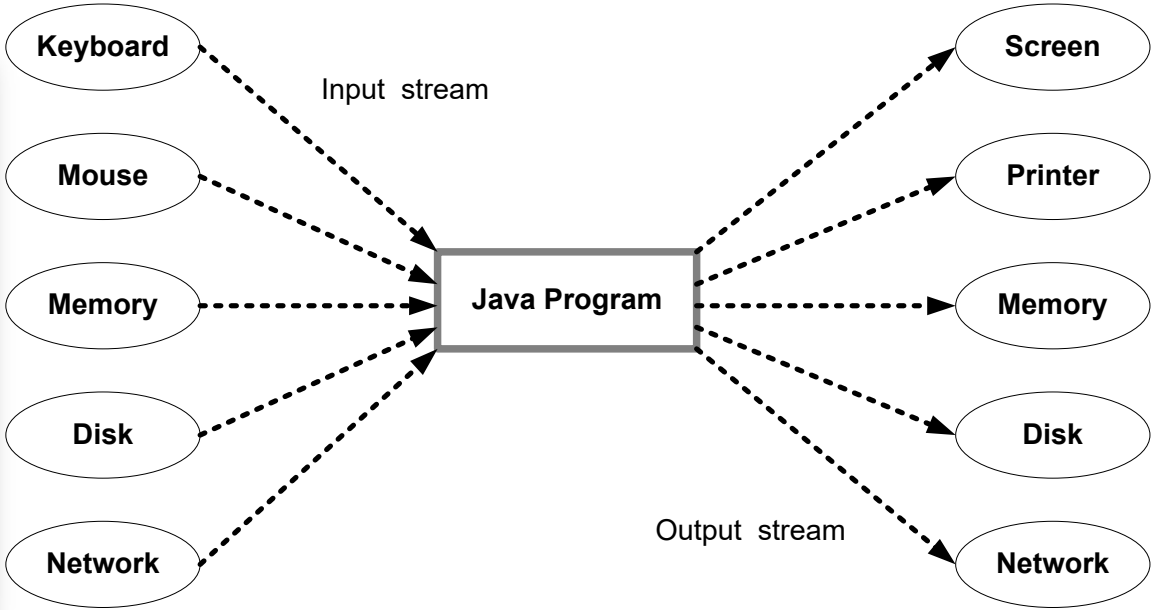
Input-Output Streams in Java

Stream in Java

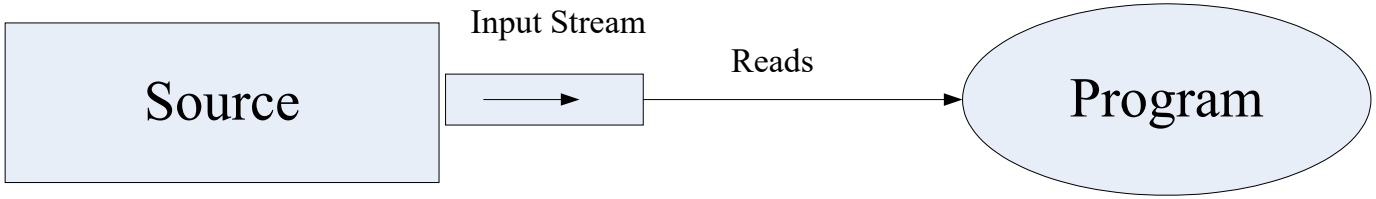
Java treats flow of data as stream.

Java streams are classified into two basic types, namely, input stream and output stream.

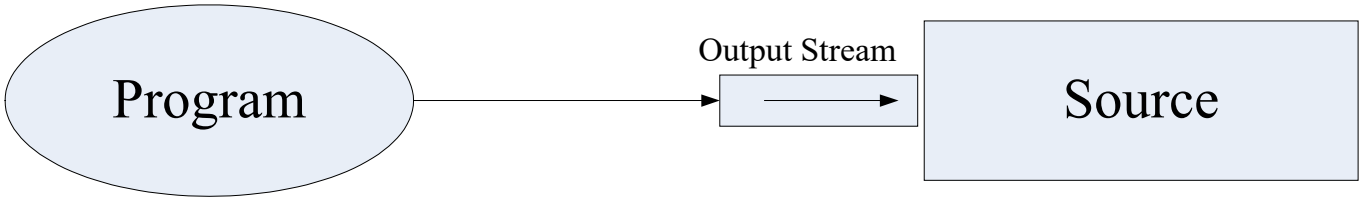
The java.io package contains a large number of stream classes to support the streams.



Input and output streams



(a) Reading data into a program



(b) Writing data to a destination

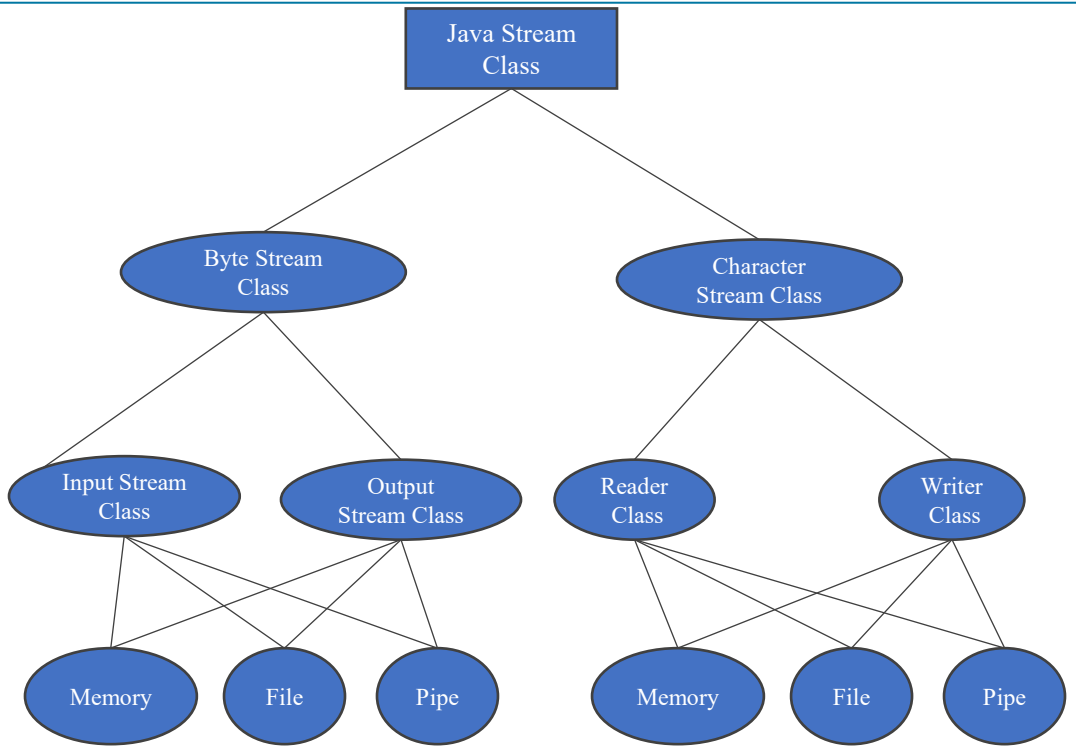
Java Classes for I-O Streams

I-O stream classes in Java

Java provides java.io package which contains a large number of stream classes to process all types of data

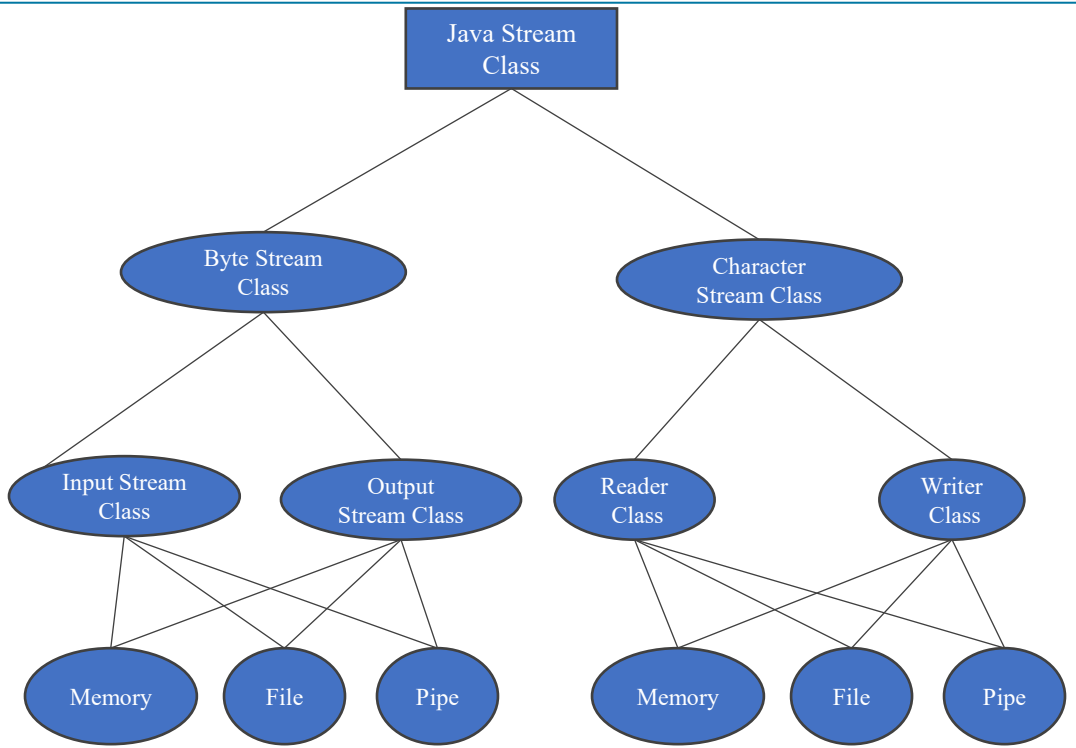
- Byte stream classes
  - Support for handling I/O operations on bytes
- Character stream classes
  - Supports for handling I/O operations on characters

Taxonomy: Java stream classes

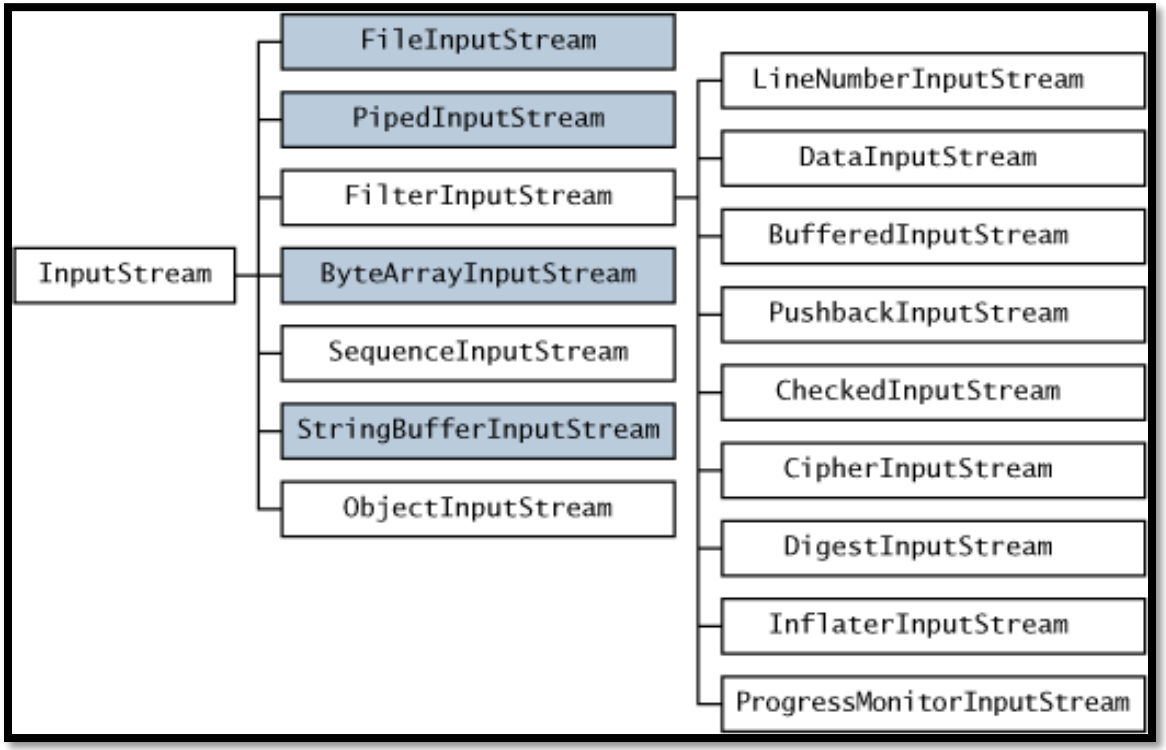


Java Input Stream Classes

Taxonomy: Java stream classes



Java input streams classes



Java input stream classes

InputStream classes is used to read 8-bit bytes and supports a number of input-related methods

- Reading bytes
- Closing streams
- Marking positions in streams
- Skipping ahead in streams
- Finding the number of bytes in stream
- and many more…

Some input stream methods

Method	Description
`read( )`	Read a byte from the input stream
`read(byte b[ ])`	Read an array of bytes into b
read(byte b[ ], int n, int m)	Reads m bytes into b starting from n^{th} byte
`available( )`	Gives number of bytes available in the input
`skip(n)`	Skips over n bytes from the input stream
`reset( )`	Goes back to the beginning of the stream
`close( )`	Close the input steam

Example:

DataInputStream

readShort( )	**readDouble( )**
readInt( )	readLine( )
readLong( )	readChar( )
readFloat( )	readBoolean( )
readUTF( )	

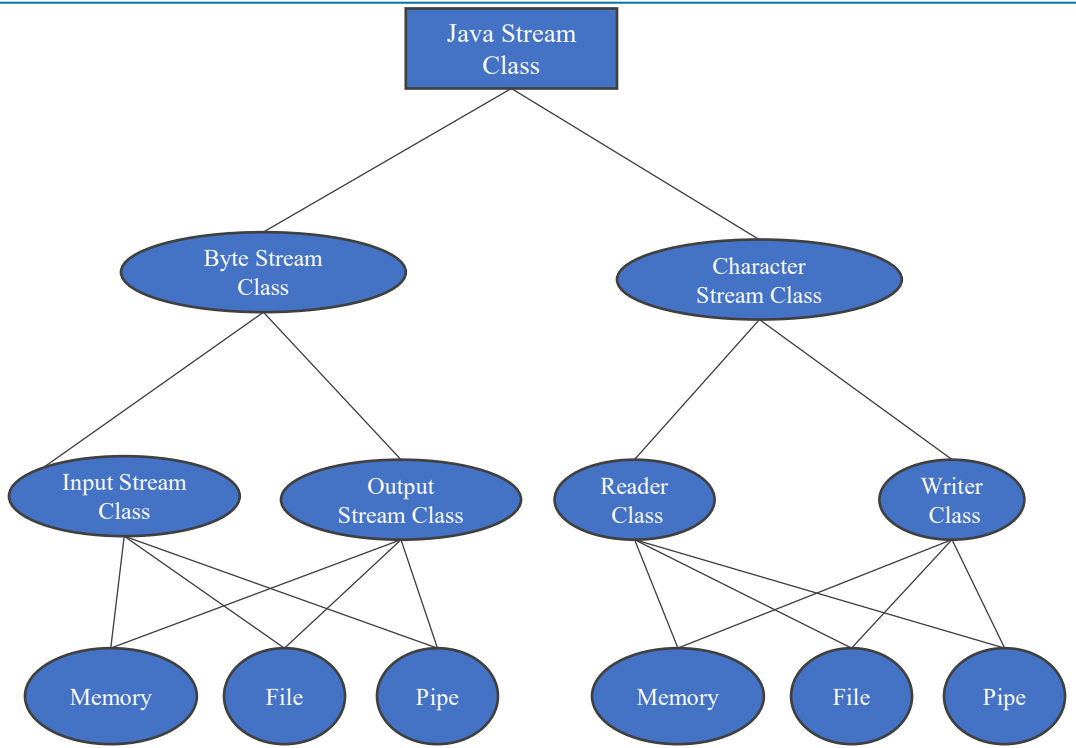
Example: Use of class InputStream

- Reading bytes
  - `int read()`
  - `int read (byte b[])`
  - `int read (byte b[], int off, int len)`
- Closing streams
  - `void close()`
- Finding the number of bytes in a stream
  - `int available()`

- Skipping ahead in a stream
  `long skip (long n)`
- Marking positions in a stream
  `void mark (int limit)`
  `void reset()`
  `boolean markSupported()`

Java Output Stream Classes

Taxonomy: Java stream classes

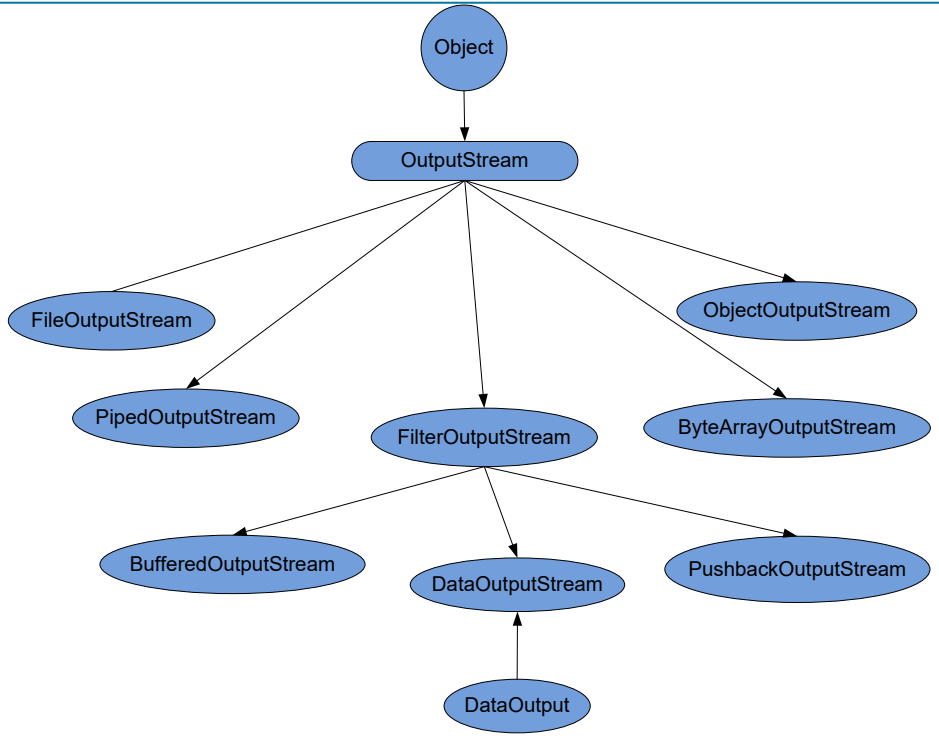


Java output stream classes

OutputStream classes is used to write 8-bit bytes and supports a number of input-related methods

- Writing bytes
- Closing streams
- Flushing streams
- etc.

Java output stream classes



Some methods in output stream classes

Method	Description
`write ( )`	Write a byte from the input stream
`write (byte b[ ])`	Write all bytes in the array b to the output steam
write (byte b[ ], int n, int m)	Write m bytes from array b starting from n^{th} byte
`close( )`	Close the output stream
`flush( )`	Flushes the output stream

Example:

DataOutputStream

`writeShort( )`	`writeDouble( )`
`writeInt( )`	`writeLine( )`
`writeLong( )`	`writeChar( )`
`writeFloat( )`	`WriteBoolean( )`
`writeUTF( )`	

Use of class OutputStream

Writing bytes

- void write (byte b)
- void write (byte b[])
- void write (byte b[], int off, int len)

Closing a stream

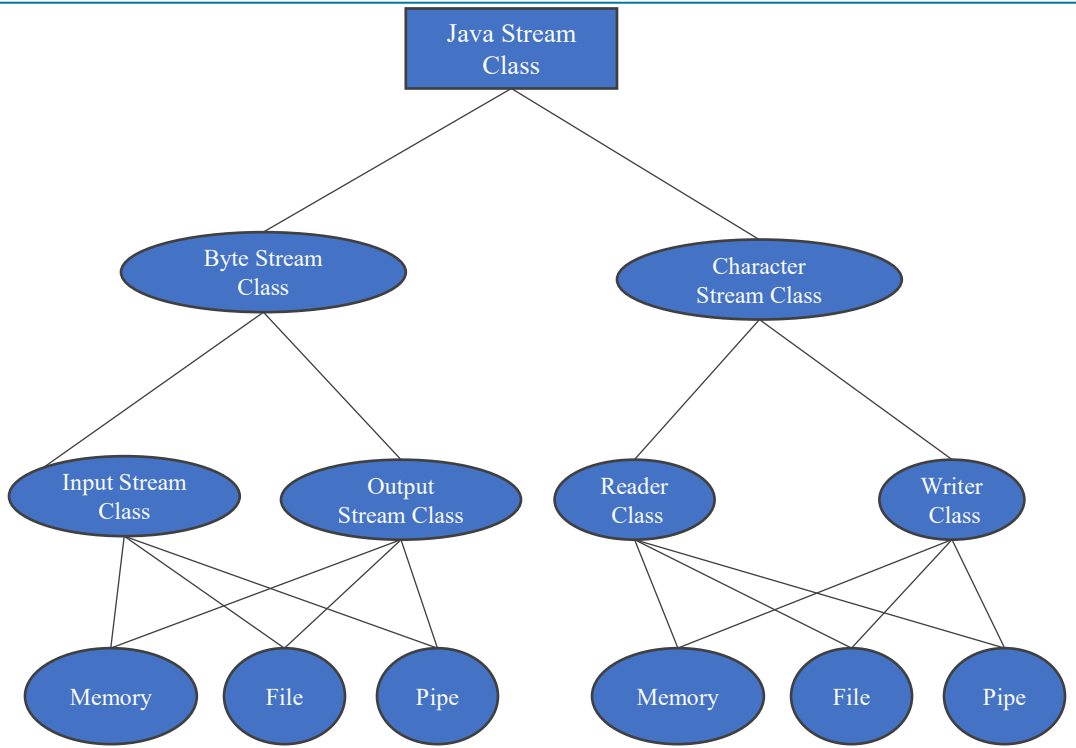
- void close()

Clearing a buffer

- void flush()

Character Stream Classes

Taxonomy: Java stream classes

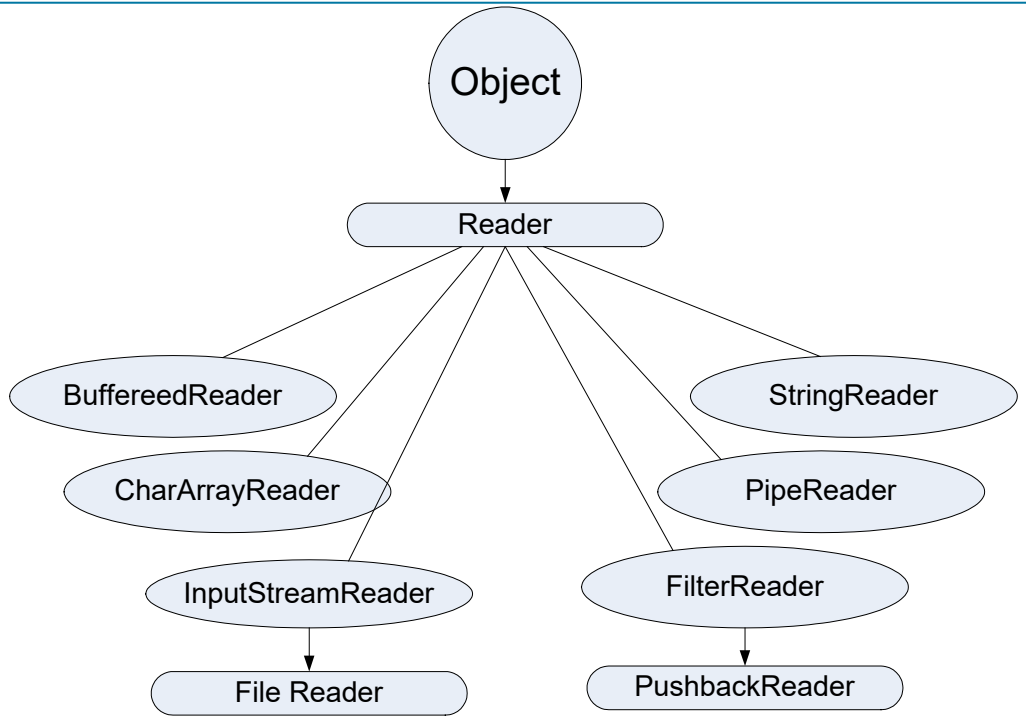


Character stream classes

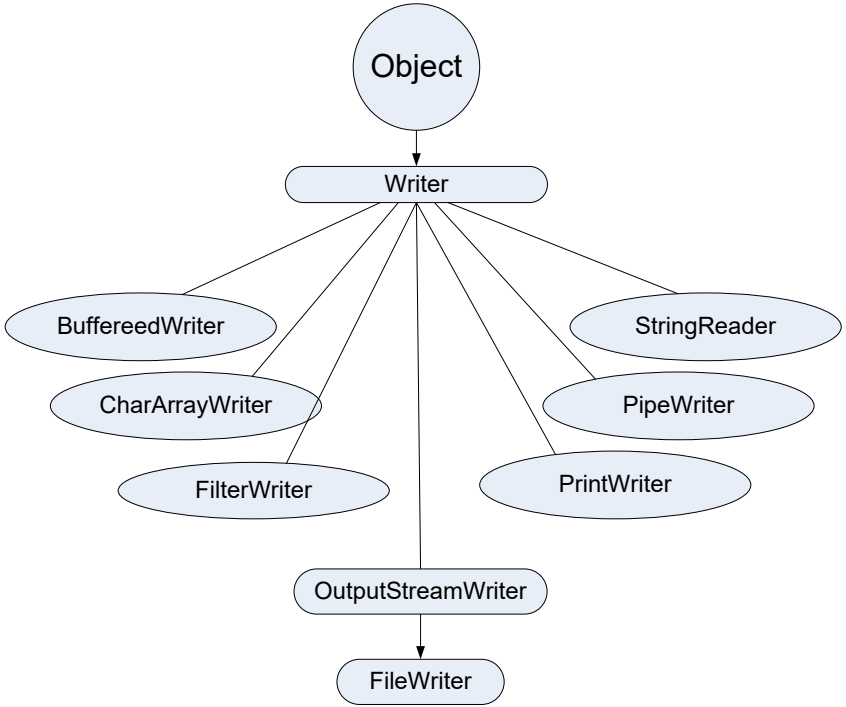
Character stream classes is used to read and write characters and supports a number of input-output related methods

- Reader stream classes
  - To read characters from files.
  - In many way, identical to InputStream classes.
- Writer stream classes
  - To write characters into files.
  - In many way, identical to OutputStream classes.

Reader stream classes



Writer stream classes



List of Important tasks and their Classes

Task	Character Stream Class	Byte Stream Class
Performing input operations	Reader	InputStream
Buffering input	BufferedReader	BufferedInputStream
Keeping track of line numbers	LineNumberReader	LineNumberInputStream
Reading from an array	CharArrayReader	ByteArrayInputStream
Translating byte stream into a character stream	InputStreamReader	(none)
Reading from files	FileReader	FileInputStream
Filtering the input	FilterReader	FilterInputStream
Pushing back characters/bytes	PushbackReader	PushbackInputStream
Reading from a pipe	PipedReader	PipedInputStream
Reading from a string	StringReader	StringBufferInputStream
Reading primitive types	(none)	DataInputStream
Performing output operations	Writer	OutputStream
Buffering output	BufferedWriter	BufferedOutputStream
Writing to an array	CharArrayWriter	ByteArrayOutputStream
Filtering the output	FilterWriter	FilterOutputStream
Translating character stream into a byte stream	OutputStreamWriter	(none)
Writing to a file	FileWriter	FileOutputStream
Printing values and objects	PrintWriter	PrintStream
Writing to a pipe	PipedWriter	PipedOutputStream
Writing to a string	StringWriter	(none)
Writing primitive types	(none)	DataOutputStream


Thank You

OBJECT ORIENTED PROGRAMMING WITH JAVA

Input-Output Handling in Java – II

Debasis Samanta
Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Usage of I-O Stream Classes

Get input using DataInputStream class

Calculator Program

```
import java.io.*;
    class InterestCalculator
    {
            public static void main(String args[ ] )
            {
                Float principalAmount = new Float(0);
                Float rateOfInterest = new Float(0);
                int numberOfYears = 0;

                DataInputStream in = new
                DataInputStream(System.in);
                String tempString;
                System.out.print("Enter Principal
                Amount: ");
                System.out.flush();
                tempString = in.readLine();
                principalAmount =
                Float.valueOf(tempString);
                System.out.print("Enter Rate of
                Interest: ");
                System.out.flush();
                tempString = in.readLine();
                rateOfInterest =
                Float.valueOf(tempString);
                System.out.print("Enter Number of Years:");
                System.out.flush();
                tempString = in.readLine();
                numberOfYears =
                Integer.parseInt(tempString);
                 // Input is over: calculate the interest
                int interestTotal =
                principalAmount*rateOfInterest*numberOfYears;
                System.out.println("Total Interest = " +
                interestTotal);
            }
    }
```

Files Handling in Java

Java File I/O

Java provides java.io package which includes numerous class definitions and methods to manipulate file and flow of data (called File I/O streams)

There are four major classes:

- File
- FileInputStream
- FileOutputStream
- RandomAccessFile

Using class File

Opening a File object

- There are three constructors
  - Way 1:
    - `File myFile;`
    - `myFile = new File(fileName);` // Constructor 1
  - Way 2:
    - `File myFile;`
    - `myFile = new File (pathName, filename);` // Constructor 2
  - Way 3:
    - `File myFile;`
    - `File myFile = new File(myDir, fileName);` // Constructor 3

Using class File

Dealing with file names

- String getName()
- String getPath()
- String getAbsolutePath()
- String getParent()
- boolean renameTo(File newFilename)

Using class File

Testing a file

- boolean exists()
- boolean canWrite()
- boolean canRead()
- boolean isFile()
- boolean isDirectory()
- boolean isAbsolute()

Using class File

Getting file information

- `long lastModified()`
- `long length()`
- `boolean delete()`

Directory utilities

- `boolean mkDir(File newDir)`
- `boolean mkDirs(File newDir)`
- `String [] list()`

Checking Status of a File

Using class File: An example

```
import java.io.File
class FileTest  {
   public static void main (String args [ ] ) throws IOException {
     File  fileToCheck;
     if (args.length > 0 ) {
        for (int i = 0; i < args.length;i++ ) {
               fileToCheck = new File(args[ i ]);
               getPaths(fileToCheck);
               getInfo(fileToCheck);
        }
     }
     else
          System.out.println (" Usage : Java file test <filename (s)

    }
```

```
public static void getPaths (File f ) throws IOException {
                    System.out.println ("Name : " + f.
getName( ) );
                    System.out.println ("Path : " + f.
getPath ( ) );
                    System.out.println ("Parent : " +
f.getParent ( ) );
    }
    public static void getInfo (File f ) throws IOException  {
                    if (f.exists ) {
                        System.out.print ("File exists ");
                        System.out.println (f.canRead( ) ?
"and is readable" : "");
                        System.out.println ( f.canWrite( )
? "and is writable" : "");
                        System.out.println ("File is last
modified : + f.lastModified( ));
                        System.out.println ("File is " +
f.length( ) + "bytes" );
                    }
                    else
                        System.err.println (" File does not
exist." );
    }
}
```

Reading a File

Example: class FileInputStream

```
class InputStreamTest
   public static void r
      int size;
    // To open a file
    FileInputStream fi
    fin = new FileInput
    size = fin.availabl
     // returns the nu
    System.out.println
    System.out.println
    for (int i = 0; i
                System
        }
```

```
System.out.println (" Remaining bytes :" + fin.available( ) );
        System.out.println ("Next ¼ is displayed : Using read( b[ ])");
        byte b[] = new byte[size/4];
        if (fin.read (b) != b.length )
          System.err.println ("File reading error : ");
        else {
            String temp = new String (b, 0, 0, b.length );
                 // Convert the bytes into string
            System.out.println (temp) ;
                 // display text string.
            System.out.println (" Still available:"+fin.available( ) );
            System.out.println (" skipping ¼ : Using skip ( )" );
            fin.skip(size/4);
            System.out.println (" File remaining for read ::"
            +fin.available( ) );
        }
      fin.close ( );          // Close the input stream
   }
}
```

Writing into a File

Example: Writing bytes into file

```
import java.io.*;
class WriteBytes {
       public static void main(String args[]) {
          cities[]={'D','E','L','H','I','\n','M','A','D','R','A','S','\n','L','O','N','D','O',
          'N','\n'}; //Declare and initialize a byte array byte
          FileOutputStream outfile=null; //create an output file stream
          try  {
                  outfile = new FileOutputStream("city.txt");
                   // Connect the outfile stream to "city.txt"
                  outfile.write(cities); //Write data to the stream
                  outfile.close();
          }
          catch(IOException ioe)    {
                System.out.println(ioe);
                System.exit(-1);
          }
       }
}
```

Reading from a File

Example: Reading bytes from file

```
import java.io.*;
class ReadBytes {
      public static void main (String args[])  {
                  FileInputStream infile = null; // Create an input file stream
                  int b;
                  try  {
                        infile = new FileInputStream(args[0]);
                              // Connect infile stream to the required file
                        while((b = infile.read()) != -1)  {
                              System.out.print((char)b); // Read and display data
                        }
                        infile.close();
                  }
                  catch(IOException ioe)  {
                              System.out.println(ioe);
                  }
      }
}
```

Copy a File into Other File
(CharacterStream Class)

Example: Reading/ writing characters

```
//Copying characters from one file into another
import java.io.*;
class CopyCharacters
{
      public static void main (String args[])
      {
      //Declare and create input and output files
      File inFile = new File("input.dat");
      File outFile = new File("output.dat");
      FileReader ins = null; // Creates file stream ins
      FileWriter outs = null;
      // Creates file stream outs
      try {
      ins = new FileReader (inFile) ;
      // Opens inFile
      outs = new FileWriter (outFile) ;
      // Opens outFile
      int ch; // Read and write till the end
```

```
while ((ch = ins.read()) != -1)
      {
            outs.write(ch) ;
      }
}
catch(IOException e) {
      System.out.println(e);
      System.exit(-1);
}
finally           //Close files
{
      try {
            ins.close();
            outs.close();
      }
      catch (IOException e) { }
      }
   } // main
} // class
```

Copying a File into Other File
(ByteStream Class)

Example: Copying bytes from one file to another

```
catch(FileNotFoundException e) {
      System.out.println("File not
      found");
    }
    catch(IOException e) {
      System.out.println(e.
      getMessage());
    }
   finally        //Close files
   {
        try {
            infile.close();
            outfile.close();
      }
        catch(IOException e){}
    }
  }
}
```

Storing Data into a File

Example: Storing and reading data

```
import java.io.*;
class ReadWritePrimitive
{
   public static void main (String args[]) throws IOException
   {
     File primitive = new File("prim.dat");
     FileOutputStream fos = new FileOutputStream(primitive);
     DataOutputStream dos = new DataOutputStream(fos);

     //Write primitive data to the "prim.dat"file
     dos.WriteInt(1999);
     dos.WriteDouble(375.85);
     dos.WriteBoolean(false);
     dos.WriteChar('X');
```

```
      dos.close();
      fos.close();
      //Read data from the "prim.dat" file
      FileInputStream fis = new FileInputStream(primitive);
      DataInputStream dis = new DataInputStream(fis);
      System.out.println(dis.ReadInt());
      System.out.println(dis.ReadDouble());
      System.out.println(dis.ReadBoolean());
      System.out.println(dis.ReadChar());

      dis.close();
      fis.close();
   }
}
```

Example: Storing and reading data in same file

```
        try {
             dis = new DataInputStream(new
             FileInputStream(intFile));     //Create input stream for intFile file
               for(int i=0;i<20;i++) {
                    int n = dis.readInt();
                    System.out.print(n + " "); }    }
                  catch(IOException ioe) {
                  System.out.println(ioe.
                  getMessage());
           }
           finally {
                  try {
                           dis.close();
                   }
                  catch(IOException ioe){ }
           }
          }
        }
```

Merging two Files into a File

Example: Concatenation and buffering

```
import java.io.*;
class SequenceBuffer
{
        public static void main (String args[]) throws
        IOException
        {
        //Declare file streams
        FileInputStream file1 = null;
        FileInputStream file2 = null;


        SequenceInputStream file3 = null;
         //Declare file3 to store combined files
        file1 = new FileInputStream("text1.dat");
        //Open the files to be concatenated
        file2 = new FileInputStream("text2.dat");
        //Open the files to be concatenated
        file3 = new SequenceInputStream(file1,file2) ;
        //Concatenate file1 and file2
```

```
        //Create buffered input and output streams
              BufferedInputStream inBuffer = new
              BufferedInputStream(file3);
              BufferedOutputStream outBuffer = new
              BufferedOutputStream(System.out);
        //Read and wri te till the end of buffers
              int ch;
              while((ch = inBuffer.read()) != -1)
                          outBuffer.write((char)ch);
                          inBuffer.close();
                          outBuffer.close();
                          file1.close();
                          file2.close();
              }
        }
```

Thank You

OBJECT ORIENTED PROGRAMMING WITH JAVA

Input-Output Handling in Java - III

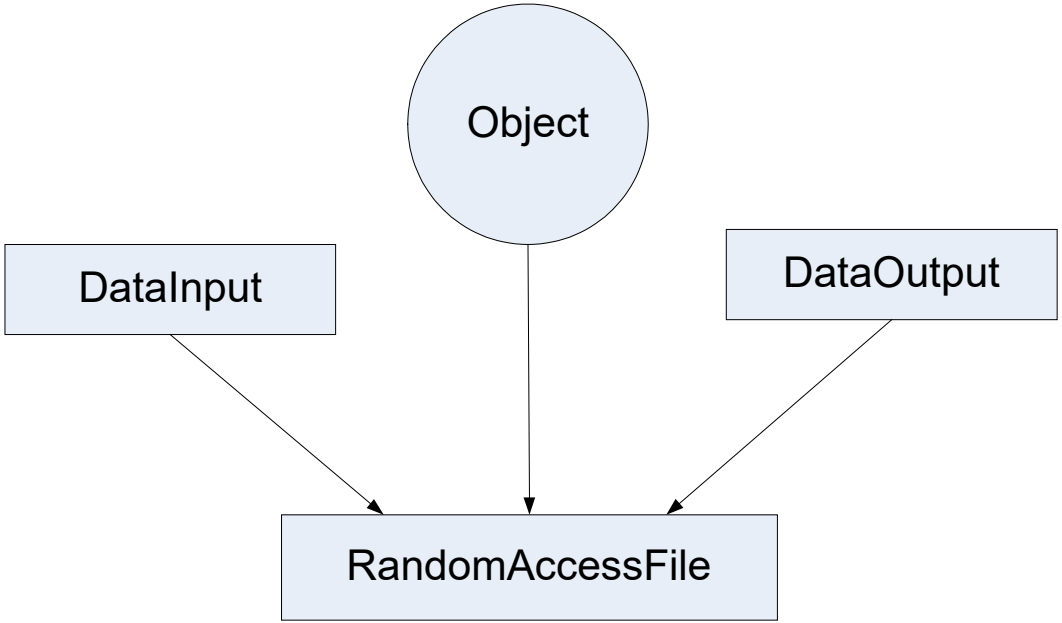
Debasis Samanta
Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Random Access Files in Java

Use of class RandomAccessFile

- As the name implies the class RandomAccessFile allows us to handle a file randomly in contrast to sequentially in InputStream or OutputStream classes
- It allows to move file pointer randomly
- Moreover, it allows read or write or read-write simultaneously.

Class: RandomAccessFile



Example: RandomAccessFile

```
import java.io.*;
class RandomIO
{
      public static void main (String args[])
      {
         RandomAccessFile file = null;
         try {
               file = new
               RandomAccessFile("rand.dat","rw");
              // Writing to the file
             file.writeChar('X');
            file.writeInt(555);
           file.writeDouble(3.1412);
            file.seek (0);
          // Go to the beginning
          // Reading from the file
         System.out.println(file.readChar());
         System.out.println(file.readInt());
         System.out.println(file.readDouble());
         file.seek(2); // Go to the second item
         System.out.println(file.readInt());
         // Go to the end and append false to
         the file
         file.seek(file.length());
         file.writeBoolean(false);
         file. seek (4) ;
         System.out.println(file.readBoolean());
                  file.close();
                  }
                  catch(IOException e)
                  {

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

Example: Appending to a RAF

```
import java.io.*;
class RandomAccess
{
      static public void main(String args[])
      {
                  RandomAccessFile rFile;
                  try
                  {
                              rFile = new RandomAccessFile("city.txt","rw");

                              rFile.seek(rFile.lenght());  // Go to the end
                              rFile.writeByte("MUMBAI\n"); //Append MUMBAI
                              rFile.close();
                  }
                  catch(IOException ioe)
                  {
                              System.out.println(ioe);
                  }
      }
}
```

Interactive Input-Output

Interactive input and output

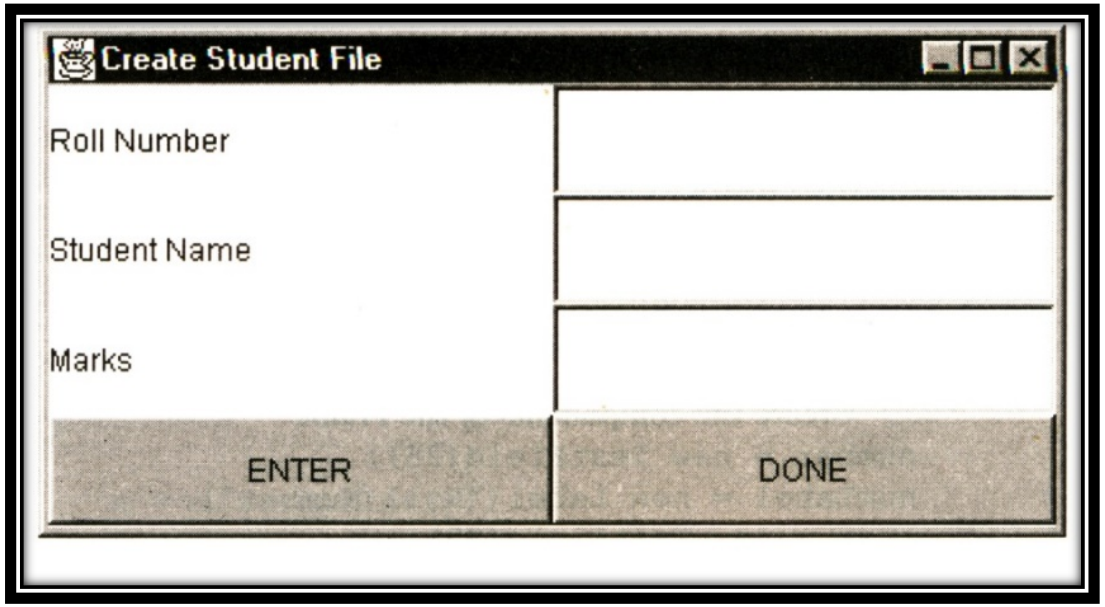
```
import java. util. *; // For usi
import java.io.*;
class Inventory {
      static DataInputStream din
      static StringTokenizer st;
      public static void main (St
      IOException
      {
      DataOutputStream dos = new
      FileOutputStream("invent.da
      // Reading from console
      System.out.println("Enter c
      st = new StringTokenizer(di
      int code = Integer.parseInt
      System.out.println("Enter n
      st = new StringTokenizer(di
      int items = Integer.parseIn
      System.out.println("Enter c
      st = new StringTokenizer(di
      double cost = new      Doubl
```

```
// Writing to the file "invent.dat"
        dos.writeInt(code);
        dos.writeInt(items);
        dos.writeDouble(cost);
        dos.close();
        // Processing data from the file
        DataInputStream dis=new DataInputStream(new
        FileInputStream("invent.dat"));
        int codeNumber = dis.readInt();
        int totalItems = dis.readInt();
        double itemCost = dis.readDouble();
        double totalCost = total Items * itemCost;
        dis.close();
        // Writing to console
        System.out.println();
        System.out.println("Code
        System.out.println("Item
        System.out.println("Total
        System.out.println("Total
        }
}
```


```
// Writing to the file "invent.dat"
        dos.writeInt(code);
        dos.writeInt(items);
        dos.writeDouble(cost);
        dos.close();
        // Processing data from the file
        DataInputStream dis=new DataInputStream(new
        FileInputStream("invent.dat"));
        int codeNumber = dis.readInt();
        int totalItems = dis.readInt();
        double itemCost = dis.readDouble();
        double totalCost = total Items * itemCost;
        dis.close();
        // Writing to console
        System.out.println();
        System.out.println("Code Number : " + codeNumber);
        System.out.println("Item Cost : " + itemCost);
        System.out.println("Total Items : " + totalItems);
        System.out.println("Total Cost  : " + totalCost);
        }
}
```

Graphical Input-Output

Graphical input and output



Graphical input and output

```
import java.io.*;
import java.awt.*;
class StudentFile extends Frame
{
	// Defining window components
	TextField number, name, marks;
	Button enter, done;
	Label numLabel, nameLabel, markLabel;
	DataOutputStream dos;

	// Initialize the Frame
	public StudentFile()
	{
			super("Create Student File");
	}
	// Setup the window
	public void setup()
	{
			resize(400, 200);
			setLayout(new GridLayout(4,2));
	// Create the components of the Frame
```

```
		number = new TextField(25);
		number = new TextField(25);
		numLabel = new Label("Roll Number");
		name = new TextField(25);
		nameLabel = new Label ("Student Name");
		marks = new TextField(25);
		markLabel = new Label("Marks");
		enter = new Button("ENTER");
		done = new Button("DONE");
		// Add the components to the Frame
		add(numLabel);
		add(number);
		add(nameLabel);
		add(name);
		add(markLabel);
		add(marks);
		add(enter);
		add(done);
		// Show the Frame
		show();
```

Graphical input and output

```
        // Open the file
        try         {
        dos new DataOutputStream( new
        FileOutputStream("student.dat"));
        }
        catch(IOException e)     {
        System.err.println(e.toString());
        System.exit(1);
        }
    }
     // Write to the file
    public void addRecord() {
    int num;
    Double d;
    num = (new
    Integer(number.getText())).intValue();
    try {
    dos.writeInt(num);
    dos.writeUTF(name.getText());
    d = new Double(marks.getText());
    dos.writeDouble(d.doubleValue());
    }
```

```
    catch(IOException e) { }
        // Clear the text fields
        number.setText(" ");
        name.setText(" ");
        marks.setText(" ");
        }
 // Adding the record and clearing the
    TextFields
      public void cleanup()
      {
      if(!number.getText(). equals(" ")) {
                addRecord();
      }
                try {
                     dos.flush();
                     dos.close();
                }
                catch(IOException e) { }
      }
```

Graphical input and output

```
// Processing the event
public boolean action(Event
event,Object o)
{
    if(event.target instanceof
       Button)
     if(event.arg.equals("ENTER")) {
          addRecord();
          return true;
     }
     return super.action(event, o);
        }
public boolean handleEvent(Event
event)
     {
```

```
    if(event.target instanceof Button)
         if(event.arg.equals("DONE")) {
                  cleanup();
                           System.exit(0);
                           return true;
         }
         return super.handleEvent(event);
         }
         // Execute the program
         public static void main (String args[])
         {
         StudentFile student = new StudentFile();
         student.setup();
         }
}
```

Another graphical Input/Output

```
import java.io.*;
import java.awt.*;
class ReadStudentFile extends Frame
{
            // Defining window components
            TextField number, name, marks;
            Button next, done;
            Label numLabel, nameLabel, markLabel;
            DataInputStream dis;
            boolean moreRecords = true;

            // Initialize the Frame
            public ReadStudentFile()
            {
            // Setup the window
            public void setup()
            {
                        resize(400,200);
                        setLayout(new
            GridLayout(4,2));
            // Create the components of the Frame
```

```
                number = new TextField(25);
                numLabel = new Label ("Roll Number");
                name = new TextField(25);
                nameLabel = new Label ("Student Name");
                marks = new TextField(25);
                markLabel = new Label("Marks");
                next new Button("NEXT");
                done = new Button("DONE");
                // Add the components to the Frame
                add(numLabel);
                add(number);
                add(nameLabel);
                add(name);
                add(markLabel);
                add(marks);
                add(next);
                add(done);
                // Show the Frame
                show();
                 // Open the file
```

Another graphical Input/Output

```
try  {
      dis new DataInputStream(new
      FileInputStream("student.dat")
                  }
                  catch(IOException e)
                  {
      System.err.println(e.toString());
      System.exit(1);
                  }
      }
        // Read from the file
      public void readRecord()
      {
       int n;
       String s;
       double d;
                  try {
                        n = dis.readInt();
                        s = dis.readUTF();
                        d dis.readDouble();
                        number.setText(String.valueOf(n));
                        name.setText(String.valueOf(s));
                        marks.setText(String.valueOf(d));
                        }
                        catch(IOException ioe) {
                        System.out.println("IO ErrorN);
                        System.exit(l);
                        }
                  }
                    // Closing the input file
                    public void cleanup()
                    {
                              try
                              {
                                        dis.close();
                              }
                              catch(IOException e) { }
                    }
```

Another graphical Input/Output

```
// Processing the event
public boolean action(Event event,Object o)
{
            if(event.target instanceof Button)
                        if(event.arg.equals("NEXT"))
                                    readRecord();
            return true;
}
public boolean handleEvent(Event event)
{
    if(event.target instanceof Button)
    if (event.arg.equals ("DONE") || moreRecords ==
            false){
                        cleanup();
                        System.exit(0);
                        return true;
                        }
            return super.handleEvent(event);
}
```

```
// Execute the program
public static void main (String
args[])
      {
      ReadStudentFile student =
      new ReadStudentFile();
      student.setup() ;
      }
}
```

Question to think…

- How Java helps programmers to develop GUIs?
- How one can develop programs like Google?


Thank You