

THE JAVA.IO PACKAGE

SESSION OBJECTIVES

At the end of this session, you will be able to -

- Discuss the concept of streams
- Discuss applets and I/O
- Discuss the standard input/output streams
- Explain the classes InputStream and OutputStream
- Describe the Byte array I/O
- Discuss Filtered and Buffered I/O operations
- Explore the class RandomAccessFile
- Describe reader and writer classes

11.1 Introduction

In the previous session we explored the java.util package in depth. We discussed the Collection classes and interfaces; the Legacy classes as well as basic utility classes such as Calendar, Date etc.

This session discusses the java.io package in detail. The java.io package contains classes and interfaces that are relevant to input/output operations. Using these classes and interfaces we can store data that may be either in the form of primitives or objects.

11.2 Applets and File I/O

Java is commonly used to create applet-based programs that are intended for the Internet or the Web. Because applets are downloaded on the client's system, they can be the cause of potential attacks such as those that corrupt data; deny access to services also called denial of service attacks, or those that simply annoy users. Hence applets must be restricted in their actions.

The Java sandbox concept was designed to impose strict controls on what certain kinds of Java programs can do or cannot do. This led to the sandbox theory wherein applets reside within a sandbox and are allowed to manipulate data only within the specified area on the hard disk. For instance, applets are not allowed to work with file related operations such as reading or writing to a file.

Keeping this concept in mind, we shall be discussing the java.io package with respect to applications only.

Java 2 has improved the default sandbox policy and now allows users to give digitally signed applet access to local resources.

11.3 Streams

A stream is a water body that carries water as it flows from one point to another. In programming terminology, a stream can be a continuous group of data or a channel through which data travels from one point to another. An input stream receives data from a source into a program and an output stream sends data to a destination from the program.

The standard input/output stream in Java is represented by 3 members of the System class: **in**, **out** and **err**. These represent the major byte streams provided by Java.

System.in: The standard input stream is used for reading characters of data. This stream responds to keyboard input or any other input source specified by the host environment or user.

It has been defined as follows-

```
public static final InputStream in
```

System.out: The standard output stream is used to typically display the output on the screen or any other output medium.

It has been defined as follows-

```
public static final PrintStream out
```

System.err: This is the standard error stream. By default, this is the user's console.

Example 1 shows how basic user input and output operations can be dealt with.

Example 1

```
class BasicIO{
    public static void main(String args[])
    {
        byte b[ ]=new byte[255];
        try{
            System.out.println("Enter a line of text");
            System.in.read(b,0,255);
            System.out.println("The line typed was ");
            String str=new String(b,"Default");
            System.out.println(str);
        }
        catch(Exception e)
        {
            System.out.println("Error occurred!");
        }
    }
}
```

The objective of the code listed in Example 1 is to accept a line of input from the keyboard and print it on the screen. We accomplish this as described below.

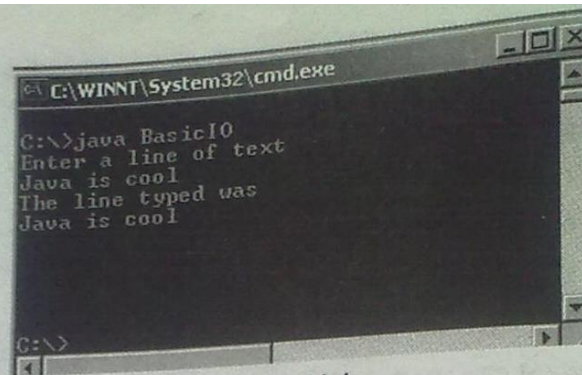
A byte array *b* that can hold 255 bytes is created in the beginning of the program. The code `System.in.read(b, 0, 255);` allows us to read 255 bytes of data from the keyboard into the byte array.

Then we create a String instance based on the byte array and print it on the screen using the following code.

```
String str=new String(b,"Default");
System.out.println(str);
```

It is likely that the operation of reading or writing may cause an exception therefore we enclose the code that performs read and write operations within a try-catch block.

The output is shown in Figure 11.1.



```
C:\WINNT\System32\cmd.exe

C:\>java BasicIO
Enter a line of text
Java is cool
The line typed was
Java is cool

C:\>
```

Figure 11.1



■ **System.in** is an instance of **InputStream**." Is this statement true or false?

11.4 The *java.io* package

Having seen the objects of **System** class that act as standard input-output streams, we now proceed towards exploring the *java.io* package.

There are 2 main categories of streams in Java:

Byte Streams : These provide a way to handle byte oriented input/output operations. They can be used with any type of object including binary data. The byte streams have the **InputStream** and **OutputStream** classes at the top of their hierarchy.

Character Streams: These provide a way to handle character oriented input/output operations. They use Unicode and can be internationalised.

Figure 11.2 shows the hierarchy of classes and interfaces within the *java.io* package.

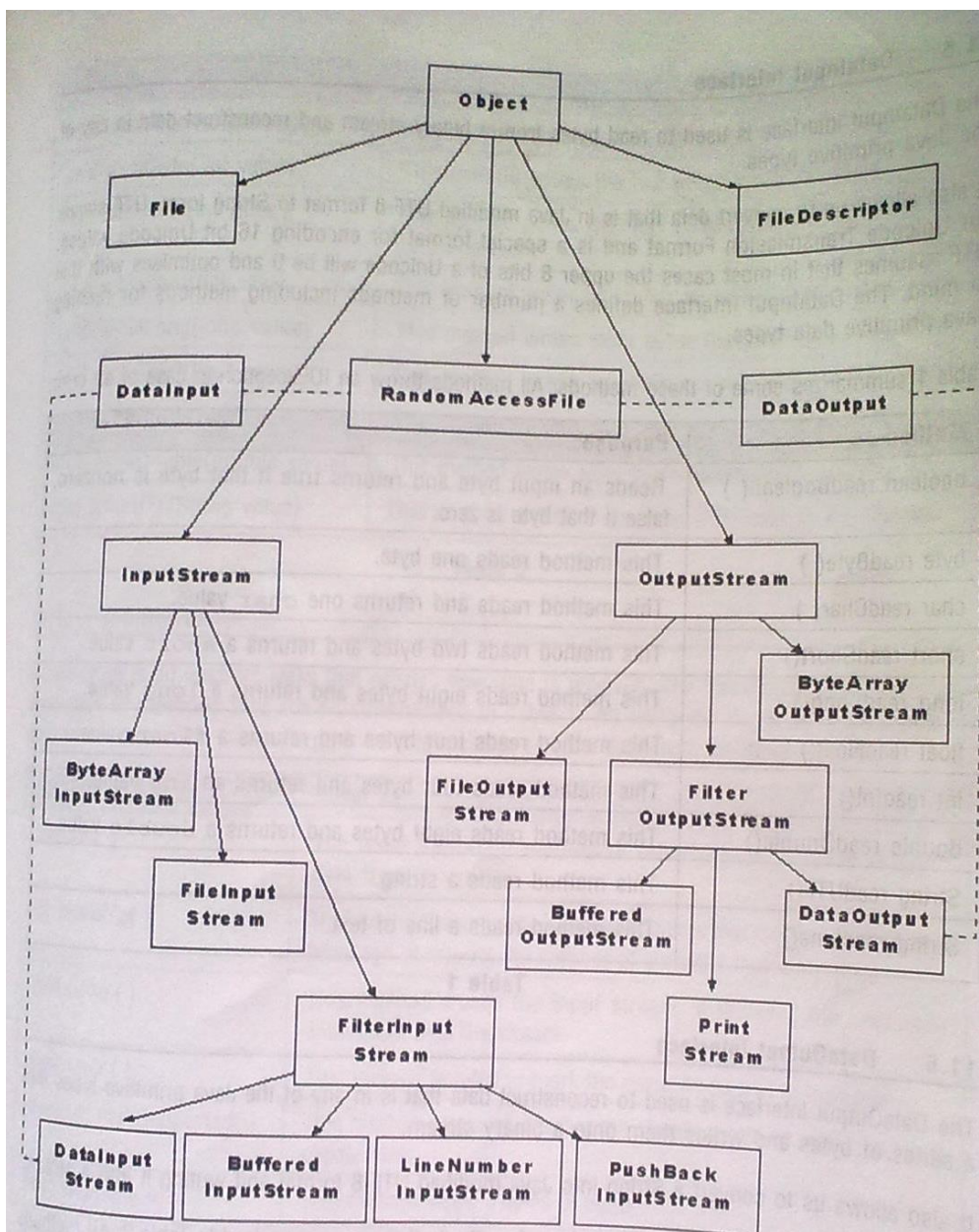


Figure 11.2

The dotted lines indicate implementation of interfaces. For instance, RandomAccessFile implements DataInput and DataOutput interfaces respectively.

11.5 DataInput interface

The DataInput interface is used to read bytes from a binary stream and reconstruct data in any of the Java primitive types.

It also allows us to convert data that is in Java modified UTF-8 format to String form. UTF stands for Unicode Transmission Format and is a special format for encoding 16 bit Unicode values. UTF assumes that in most cases the upper 8 bits of a Unicode will be 0 and optimises with that in mind. The DataInput interface defines a number of methods including methods for reading Java primitive data types.

Table 1 summarizes some of these methods. All methods throw an IOException in case of an error.

Method	Purpose
boolean readBoolean()	Reads an input byte and returns true if that byte is nonzero, false if that byte is zero.
byte readByte()	This method reads one byte.
char readChar()	This method reads and returns one char value.
short readShort()	This method reads two bytes and returns a short value.
long readLong()	This method reads eight bytes and returns a long value.
float readFloat()	This method reads four bytes and returns a float value.
int readInt()	This method reads four bytes and returns an int value.
double readDouble()	This method reads eight bytes and returns a double value.
String readUTF()	This method reads a string.
String readLine()	This method reads a line of text.

Table 1

11.6 DataOutput interface

The DataOutput interface is used to reconstruct data that is in any of the Java primitive types into a series of bytes and writes them onto a binary stream.

It also allows us to convert a String into Java modified UTF-8 format and writing it into a stream.

The DataOutput interface defines a number of methods that are summarized in Table 2. All methods throw an IOException in case of an error.

Method	Purpose
void writeBoolean(boolean b)	Writes a boolean value to the stream.
void writeByte(int value)	This method writes the low order 8 bits of <i>value</i> .
void writeChar(int value)	This method writes the char value comprised of two bytes to the stream.
void writeShort(int value)	This method writes two bytes representing the <i>short</i> value.
void writeLong(long value)	This method writes eight bytes representing a long value.
void writeFloat(float value)	This method writes four bytes representing a float value.
void writeInt(int value)	This method writes four bytes.
void writeDouble(double value)	This method writes eight bytes representing a double value.
void writeUTF(String value)	This method writes a string in UTF form to the stream.

Table 2

11.7 InputStream

InputStream is an abstract class that defines how data is received. The InputStream class provides a number of methods that are as summarized in Table 1. All methods throw an IOException in case of an error. The basic purpose of this class is to read data from an input stream.

Method	Purpose
int read()	The most important method of InputStream. It reads bytes of data from a stream.
int available()	This method returns the number of bytes that can be read without blockage. It returns the number of available bytes.
void close ()	This method closes the input stream. It releases the resources associated with the stream.
void mark ()	This method is used to mark the current position in the stream.
boolean markSupported()	This method indicates whether the stream supports mark and reset capabilities.
long skip(long n)	This method skips <i>n</i> bytes of input.

Table 3

?

What is wrong in the following code?

```
InputStream in =new InputStream("abc.txt");
```

11.8 OutputStream

The OutputStream class is also abstract and defines the way in which outputs are written to streams. The class provides a set of methods that are summarized as depicted in Table 2. These methods also throw an IOException if an error occurs. This class is used to write data to a stream.

Method	Purpose
void write()	This is the most fundamental operation in OutputStream class. This method writes a byte. It can also write an array of bytes.
void flush()	This method flushes the stream. The buffered data is written to the output stream.
void close()	This method closes the stream. It is used to release any resource associated with the stream.

Table 4

11.9 FileInputStream

This class is used to read input from a file in the form of a stream. Commonly used constructors of this class are shown below:

- `FileInputStream(String filename)` : Creates an InputStream that we can use to read bytes from a file. filename is full path name of a file.
- `FileInputStream(File name)` : Creates an InputStream that we can use to read bytes from a file. name is a File object.

Example 2 demonstrates how to use a FileInputStream object.

Example 2

```
import java.io.*;
class FileDemo{
    public static void main(String args[]) throws Exception
    {
        int size;
        InputStream f=new FileInputStream(args[0]);
        System.out.println("Bytes available to read
        :"+(size=f.available()));
        char str[]=new char[200];
```



```

for(int i=0;i<size;i++)
{
    str[i]=((char)f.read());
    System.out.print(str[i]);
}
System.out.println("");
f.close();
}
}

```

In Example 2, we create an `InputStream` that will enable us to read from the file whose name is specified in the command line. We then print the number of bytes available to read by using the `available()` method.

?

In the above code why is it necessary to use a cast for the `f.read()` statement?

The output of the example is shown in Figure 11.3

```

C:\WINNT\System32\cmd.exe
C:\>java FileDemo story.txt
Bytes available to read :127
Once upon a time there lived a king.
He was kind , generous and good-hearted.
He was extremely liked by all his citizens.
C:\>

```

Figure 11.3

11.10 `FileOutputStream`

This class is used to write output to a file stream. Its constructors are summarized below-

- `FileOutputStream(String filename)` : Creates an `OutputStream` that we can use to write bytes to a file. Here *filename* is full path name of a file.
- `FileOutputStream(File name)` : Creates an `OutputStream` that we can use to write bytes to a file. Here *name* is a `File` object.
- `FileOutputStream(String filename,boolean flag)` : Creates an `OutputStream` that we can use to write bytes to a file. *name* is a `File` object. If *flag* is true, file is opened in append mode.

The code in Example 3 shows how to create and write into a `FileOutputStream`.

Example 3

```
import java.io.*;

class FileOutputDemo{
    public static void main(String args[])
    {
        byte b[]=new byte[80];
        try{
            System.out.println("Enter a line to be saved into
a file");
            int bytes=System.in.read(b);
            FileOutputStream fo=new FileOutputStream("xyz.txt");
            fo.write(b,0,bytes);
            System.out.println("Written!");
        }
        catch(IOException e)
        {
            System.out.println("Error creating file!");
        }
    }
}
```

The output will be as shown in Figure 11.4

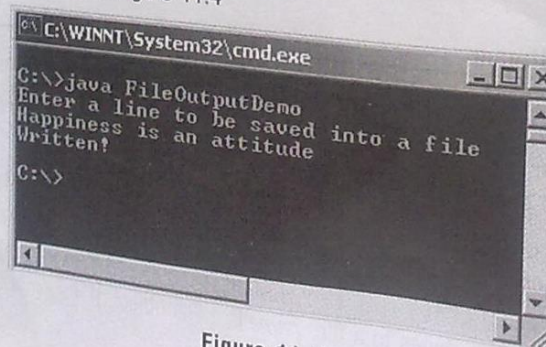


Figure 11.4



Given the code

```
new FileOutputStream("data", true);
```

What will be its result?

11.11 **ByteArrayInputStream**

This class is used to create an input stream using an array of bytes. This class does not support any new methods. It only overrides methods of `InputStream` like `read()`, `skip()`, `available()` and `reset()`.

Its constructors are shown below-

- `ByteArrayInputStream(byte b[])` : Creates a `ByteArrayInputStream` with `b` as the input source.
- `ByteArrayInputStream(byte b[], int start, int num)` : Creates a `ByteArrayInputStream` that begins with the character at `start` position and is `num` bytes long.

It implements both `mark()` and `reset()` methods.

11.12 **ByteArrayOutputStream**

This class is used to create an output stream using a byte array as the destination. It also provides additional capabilities for the output array to grow so as to accommodate the new data that is written.

This class defines two constructors. One takes an integer argument used to set the output byte array to an initial size. The second one does not take any argument and sets the output buffer to a default size. This class provides some additional methods like `toByteArray()` and `toString()` that convert the stream to a byte array or `String` object respectively.

11.13 **File**

Unlike other classes that work on streams, `File` class directly works with files on the filesystem. The files are named using the file-naming conventions of the host operating system. These conventions are encapsulated using the `File` class constants. All common file and directory operations are performed using the access methods provided by the `File` class. Methods of this class allow the creating, deleting and renaming of files, provide access to the pathname of the file, determine whether any object is a file or directory and check the read and write access permissions.

Just like access methods, the directory methods also allow creating, deleting, renaming and listing of directories. These methods allow directory trees to cross by providing access to the parent and sibling directories.

This class is used whenever we need to work with files or directories on the file system and perform operations like creating, deleting etc.

"Using delete() on a File instance will delete the file from the disk." Is this statement true or false?

Example 4 checks if given file is a directory or not by means of the File class's methods.

Example 4

```
import java.io.File;

class Filetest{
    static void p(String s)
    {
        System.out.println(s);
    }

    public static void main(String args[])
    {
        File f1=new File(args[0]);
        p(f1.getName()+(f1.exists()?" exists" : " does not exist" ));
        p("File size :"+f1.length()+"bytes");
        p("Is"+(f1.isDirectory()? " a directory": " not a directory"));
    }
}
```

The output is as shown in Figure 11.5

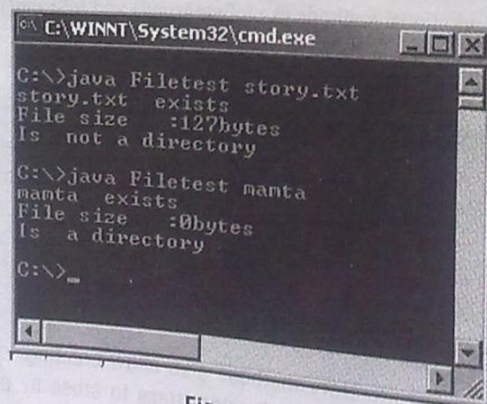


Figure 11.5

11.14 FileDescriptor

This class provides access to the file descriptors that are maintained by the operating system when files and directories are being accessed. In practical use, a file descriptor is used to create a `FileInputStream` or `FileOutputStream` to contain it. Applications should not create their own file descriptors.

11.15 Filtered Input and Output classes

The filtered input and output stream classes provided by Java help to filter I/O in a number of ways. These classes themselves do not deal with any filtering operations but instead delegate them to their sub-classes such as `BufferedInputStream` or `DataOutputStream`.

11.15.1 FilterInputStream

This class is abstract and is the parent of all filtered input stream classes.

Its constructor is –

```
public FilterInputStream(InputStream in)
```

?

"`DataInputStream` inherits directly from `FilterInputStream`." Is this statement true or false?

11.15.2 FilterOutputStream

This class is the supplement to the class `FilterInputStream`. It is the parent of all filtered output stream classes. Its constructor is–

```
public FilterOutputStream(OutputStream out)
```

11.16 Buffered I/O classes

A buffer is a temporary storage area for data. By storing the data in a buffer, we save time as we immediately get it from the buffer instead of going back to the original source of the data.

Java uses buffered input and output to temporarily cache data read from or written to a stream. This helps programs to read or write small amounts of data without adversely affecting the performance of the system.

Filters operate on the buffer which is located between the program and the destination of the buffered stream.

11.16.1 BufferedInputStream

This class defines 2 constructors-

- `BufferedInputStream (InputStream)`: Creates a buffered input stream for the specified `InputStream` instance
- `BufferedInputStream (InputStream , int)`: Creates a buffered input stream of a given size for the specified `InputStream` instance

The simplest way to read data from an instance of `BufferedInputStream` is to invoke its `read()` method.

11.16.2 BufferedOutputStream

This class performs output buffering in a manner corresponding to the class `BufferedInputStream`. This class also defines two constructors.

- `BufferedOutputStream (OutputStream)`: Creates a buffered output stream for the specified `OutputStream` instance
- `BufferedOutputStream (OutputStream , int)`: Creates a buffered output stream of a given size for the specified `OutputStream` instance

11.17 RandomAccessFile

This class does not extend either `InputStream` or `OutputStream` but instead implements the `DataInput` and `DataOutput` interfaces.

Its main advantage is its capability to perform I/O to specific locations within a file. Data can be read or written to random locations within a file instead of a continuous storage of information.

It supports reading/writing of all primitive types. For instance there are methods like `readInt()`, `writeFloat()` etc. All methods of this class throw an `IOException` in case of an error.

This class defines 2 constructors both of which take a mode ("r" or "rw") as a parameter for read only, or read/write.

```
RandomAccessFile(File file, String mode)
```

```
RandomAccessFile(String file, String mode)
```

Example 5 shows how to use a `RandomAccessFile` object to read from a file.

Example 5

```
import java.io.*;

class RandomAccessFileDemo{
    public static void main(String args[])
    {
        byte b;
        try{
            RandomAccessFile fl=new RandomAccessFile(args[0], "r");
            long size=fl.length();
            long fp=0;
            while(fp<size)
            {
                String s=fl.readLine();
                System.out.println(s);
                fp=fl.getFilePointer();
            }
        }
        catch(IOException e)
        {
            System.out.println("File does not exist!");
        }
    }
}
```

The output is as shown in Figure 11.6.

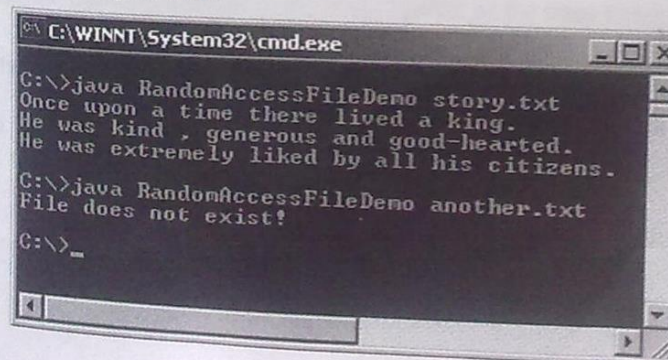


Figure 11.6



What will happen if we tried to create an instance of `RandomAccessFile` with mode "r" but the file never existed?

11.18 Character streams

As mentioned earlier, character streams provide a way to handle character oriented input/output operations. They support Unicode and can be internationalised. `Reader` and `Writer` are abstract classes at the top of the class hierarchy that supports reading and writing of Unicode character streams.

11.18.1 Reader

This class is used for reading character streams. The methods of this class are summarized in Table 5.

Method	Purpose
<code>abstract void close()</code>	Closes the stream
<code>void mark(int limit)</code>	Marks the current position in the stream
<code>boolean markSupported()</code>	Determines whether <code>mark()</code> is supported or not.
<code>int read()</code>	Reads one character
<code>int read(char buf[])</code>	Reads characters into an array
<code>abstract int read(char buf[], int offset, int length)</code>	Reads characters into a portion of an array.
<code>boolean ready()</code>	Determines if the stream is ready to be run.
<code>void reset()</code>	Resets the stream.
<code>long skip(long n)</code>	Skips n characters.

Table 5

11.18.2 Writer

This class is also abstract and supports writing into streams through methods that can be overridden by its subclasses. It includes the following methods:

Method	Purpose
abstract void close()	Closes the stream
void mark(int limit)	Marks the current position in the stream
boolean markSupported()	Determines whether mark() is supported or not.
int read()	Reads one character
int read(char buf[])	Reads characters into an array
abstract int read(char buf[], int offset, int length)	Reads characters into a portion of an array
boolean ready()	Determines if the stream is ready to be run
void reset()	Resets the stream
long skip(long n)	Skips characters

Table 6

11.19 The PrintWriter Class

The `PrintWriter` class is a character-based class that is useful for console output. This class provides support for Unicode characters. The printed output is flushed and tested for any errors using the `checkError()` method. To set an error condition, the `setError()` method is used. The `PrintWriter` class also provides support for printing primitive data types, character arrays, strings and objects.

11.20 Character Array Input Output

The `CharArrayReader` and `CharArrayWriter` classes are similar to `ByteArrayInputStream` and `ByteArrayOutputStream` in that they support input and output from memory buffers.

`CharArrayReader` and `CharArrayWriter` support 8-bit character input and output.

`CharArrayWriter` adds the following methods to the ones provided by the class `Writer`.

- `reset()` : Resets the buffer.
- `size()` : Returns the current size of the buffer.
- `toCharArray()` : Returns the character array copy of the output buffer.
- `toString()` : Converts the output buffer to a string object.
- `writeTo()` : Writes the buffer to another output stream.

The Session in Brief

- According to the sandbox theory, applets reside within a sandbox and are allowed to manipulate data only within the specified area on the hard disk
- A stream is a path travelled by data in a program.
- When a stream of data is being sent or received, we refer to it as writing and reading a stream respectively.
- The standard input-output stream consists of `System.out`, `System.in` and `System.err` streams.
- `InputStream` is an abstract class that defines how data is received.
- `InputStream` provides a number of methods for reading and taking streams of data as input.
- The `OutputStream` class is also abstract. It defines the way in which output is written to streams.
- `ByteArrayInputStream` creates an input stream from the memory buffer while `ByteArrayOutputStream` creates an output stream on a byte array.
- Java supports file input and output with the help of `File`, `FileDescriptor`, `FileInputStream` and `FileOutputStream` classes.
- The class `RandomAccessFile` provides the capability to perform I/O to specific locations within a file.
- `Reader` and `Writer` classes are abstract classes that support reading and writing of Unicode character streams.

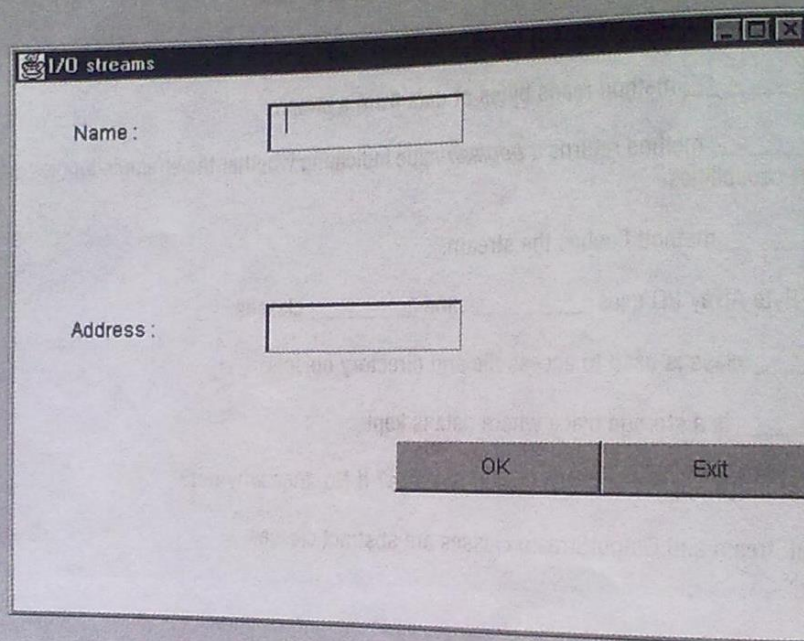
Check Your Progress

1. _____ are pipelines for sending and receiving information in Java programs.
2. While reading or writing a stream if an error occurs, an _____ is thrown.
3. _____ class defines standard input and output stream.
4. _____ is the standard error stream.
5. _____ method reads bytes of data from a stream.
6. _____ method returns a *boolean* value indicating whether the stream supports mark and reset capabilities.
7. _____ method flushes the stream.
8. The Byte Array I/O uses _____ and _____ classes.
9. _____ class is used to access file and directory objects.
10. _____ is a storage place where data is kept.
11. Can you use applets with the java.io package? If No, then why not?
12. InputStream and OutputStream classes are abstract classes.

True/False

Do it yourself

1. Write a program that checks if given file is a directory or not.
2. Write a program that uses a GUI and saves its contents to a text file. The GUI should look as shown below-



The image shows a Java Swing window titled "I/O streams". Inside the window, there are two text input fields. The first field is preceded by the label "Name :". The second field is preceded by the label "Address :". At the bottom of the window, there are two buttons: "OK" and "Exit".